TU WIEN Informatics

Efficient Annotation of Complex Documents through Active Learning for Retraining a Machine Learning Classifier

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Logic and Computation

by

B. Sc. Benedikt Häcker
Registration Number 11713128

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dr. Allan Hanbury
Assistance: Projektass. Dipl.-Ing. Tobias Fink

Vienna, 21st July, 2021

Benedikt Häcker

Allan Hanbury

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

B. Sc. Benedikt Häcker

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 21. Juli 2021

Benedikt Häcker

Acknowledgements

Diese Diplomarbeit widme ich den beiden Menschen, die mir das Studium und noch viel mehr ermöglicht haben. Danke für alles Mama & Papa.

Kurzfassung

Überwachtes Maschinelles Lernen benötigt Training mit gelabelten Daten, welche teuer sind, wenn Menschen sie annotieren müssen. Aktives Lernen zielt darauf ab, den Annotationsaufwand zu reduzieren indem es geeignete Trainingssamples auswählt welche zu einer höheren Performance des ML-Algorithmus führen als zufällig gewählte Traininsdaten. Die Aufgabe für die wir Aktives Lernen untersuchen möchten ist die Klassifikation von Dokumenten. Wir sehen uns mit einer Situation konfrontiert in der wir keinen Zugang zu den ungelabelten Daten und keinen Zugang zu dem ML-Modell haben. Die Auswahl der Samples basiert alleinig auf den Vorhesagevektoren welche von dem ML-Modell erzeugt werden. Wir experimentieren mit Szenarien in welchen wir Zugang zu Daten und Modell haben, um zu sehen ob unsere Methoden besser performen in solch einem Fall. Die Aktives Lernen Methoden, die wir verwenden, bauen auf verschiedenen Annahmen auf und können in drei Familien eingeteilt werden: Individuelle Score Berechnungen, Distanzbasierte Teilmengen Auswahl und Methoden zur Vorhersage der Modellverbesserung. Um die Aktives Lernen Methoden zu evaluieren, führen wir ein neues Maß ein und benutzen es, um die verschiedenen Methoden zu vergleichen. Unsere Experimente zeigen einen klaren Vorteil des Einsatzes von Aktives Lernen Methoden gegenüber keinem Einsatz von Aktivem Lernen.

Abstract

Supervised machine learning algorithms require training on labeled training data which is expensive to obtain when the labels have to by annotated by humans. Active Learning aims to reduce the annotation effort by selecting suitable training samples which yield a higher performance of the machine learning algorithm then random chosen training samples. The task we want to explore Active Learning methods for is the classification of documents. We face a situation where we do not have access to the unlabeled data and do not have access to the machine learning model. The selection of samples happens solely on the prediction vector made by the machine learning model for individual samples. We experiment with scenarios where we have access to data and model to see if our methods perform better in such a case. The Active Learning methods which we employ are built on different assumptions and can be categorized into three families: individual score calculations, distance based subset selections and model improvement prediction methods. To evaluate Active Learning methods we introduce a novel measure and use it to compare different methods. Our experiments show a clear advantage of using Active Learning methods over no Active Learning.

Contents

CHAPTER 1

Introduction

1.1 Motivation & Problem Statement

Machine learning is a multipurpose technology in the sense that it can be applied to many abstract and real world problems. One of the applications lies in information extraction from documents, in particular in classifying documents and segments of documents.
This master thesis is conducted within the scope of the K.Rex project [1]. The task we face here is the automatic analysis of real world documents. Documents can be emails, handwritten notes, chat records, blueprints, invoices, contracts, et cetera. The ultimate goal is to find relevant documents in the set of all documents. What a relevant document is differs from data set to data set and is not known beforehand.
In the current phase of the project the task is to classify documents into classes that are defined by the users of the system. For the classification a machine learning algorithm is used. The machine learning algorithm requires training on a batch of documents, also called samples. Different batches of training samples lead to different behavior of the trained machine learning algorithm.
We have the task of finding the right batch of documents for training the classification algorithm, such that the performance of the trained algorithm is higher then the performance of the algorithm trained on a random batch of documents. In the best case, we find the training batch that leads to the algorithm outperforming all algorithms of the same architecture trained on the other possible training batches. Performance is measured as the percentage of correctly classified documents, collected in a withheld test set which is never used for training. A high percentage of correctly classified samples means a high performance. The crucial point is that only labeled documents can be used for training the classification algorithm. A labeled document is a document that is assigned to a class. The labeling is done by human experts. Experts do not work for free, so the labeling comes with a price. Since the hourly rates of experts are high, the labeling is costly as well. With a fixed budget of human expert hours, we want to

maximize the performance of the classification algorithm. Finding the right samples for that, without knowing their labels a priori, is the problem we want to solve. The research area engaged with this is *Active Learning*.

1.2 Aim of the Work

The aim of the work is to review the state of the art methods used in Active Learning and to identify the best approaches for the given project set up. Implementation of the most promising approaches and experiments on public data sets and case specific data should clarify if Active Learning can reduce the number of needed training samples and therefore reduce the cost of labeling. We aim to adapt algorithms like Multi-Armed bandits and the concept of reinforcement learning to the active learning framework and develop new methods to select the batch of training samples, which maximize the performance of the classification algorithm. The research questions we want to answer are:

- By how much can the use of Active Learning methods reduce the number of needed training samples without reducing classification performance significantly?
- Which Active Learning method performs best?

CHAPTER 2

Related Work

In this section we describe related work in the area of Active Learning. We summarize state of the art frameworks and methods and see if they are applicable to our use-case. We further categorize the presented methods on different dimensions. If we make use of prior work in our work, we refer to the section where we do so.
Active Learning is a subfield of supervised learning, which is a subset of machine learning. Supervised learning algorithms need to be trained on a lot of labeled samples. An Active Learning algorithm is allowed to choose the instances from which it learns [2], opposed to a passive or offline learner which cannot choose the data it learns from. "The key hypothesis is that if the learning algorithm is allowed to choose the data from which it learns [...] it will perform better with less training" [2]. Being active means that the Active Learner puts out *queries* to an oracle or human annotator. The queries are sets of unlabeled samples which are then labeled by the oracle or human annotator.
From a data point of view, a rough subdivision of Active Learning methods can be made in membership query synthesis, stream-based selective sampling and pool-based sampling [2]. They differ with respect to the availability of unlabeled data.

Membership queries

Membership queries are sets of synthetic samples generated by the Active Learning method itself. So instead of selecting a set of real samples, the Active Learning method generates new samples. The human annotator is then asked to label the newly generated samples. Often these artificial samples have no natural meaning for the human annotator. In the case of images one might think of random generated pixels and in the case of text one might imagine gibberish, which a human annotator has to give meaningful labels to. Therefore this approach "is reasonable for many problems, but labeling such arbitrary instances can be awkward [...]" [2].

Stream-based

Stream-based Active Learning methods decide for individual samples drawn from a data source if the human annotator should label the sample or not. Methods from this category are usually employed when "[...] obtaining an unlabeled instance is free (or inexpensive)" [2]. The decision if a sample should be labeled by a human annotator can be made based on a simple threshold: If the machine learning models prediction confidence is under that threshold it should be labeled by a human annotator, otherwise not. While stream-based methods are straightforward and computationally cheaper than sample generating methods, they have one big drawback. Since a stream-based Active Learning method decides for every sample individually if the label should be requested or not, the number of samples that should be labeled are not known a priori. When wanting to reduce the annotation effort this type of methods may not be suitable.

Pool-based

Pool-based sampling methods overcome this issue of not knowing in advance how many samples a human annotator has to label. These approaches assume a pool of unlabeled samples, which can then be ranked or partitioned into subsets. The assumption of "large collections of unlabeled data [that] can be gathered at once" [2] holds for many real-world problems. The methods on how to rank the samples or select a subset of samples can be the same as in a stream-based approach. The decision for a stream-based or pool-based approach is mostly based on the circumstances of data gathering and management. If the freedom of choice is given, pool-based approaches are to be preferred. In the K.Rex project we deal with a collection of documents, which is the pool of unlabeled samples we just talked about. A schematic comparison of the three different scenarios can be seen in Figure 2.1.

Retraining

Another typical set up where Active Learning is used is retraining an already trained base model, which is also called *retraining-based Active Learning*. In [3] the authors retrain a machine learning model on unlabeled samples. Since labeled samples are needed to retrain a supervised machine learning algorithm, the authors do the retraining for all possible labels. Based on the outcome an average-case or worse-case performance is obtained. The sample which maximizes the performance is chosen as the Active Learning query to the oracle or human annotator. This requires k rounds of retraining the machine leaning model for a single sample, where k is the number of classes. When retraining the machine learning model is computationally expensive, this approach is not suitable. Nonetheless, the circumstances of retraining an already trained base model apply to the K.Rex project as well. The purpose of retraining the model here is that we want to adapt the base model for each case individually. The reasons for this are legal requirements, which do not allow information transfer from one case to the other. For each new case the training data is obtained by human experts, who annotate and label documents to do

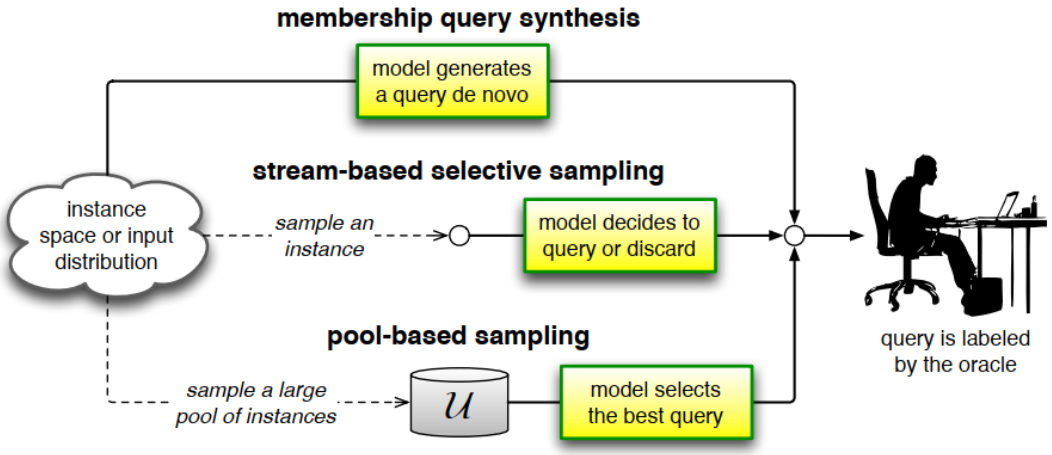


Figure 2.1: Diagram illustrating the three main active learning scenarios from [2]

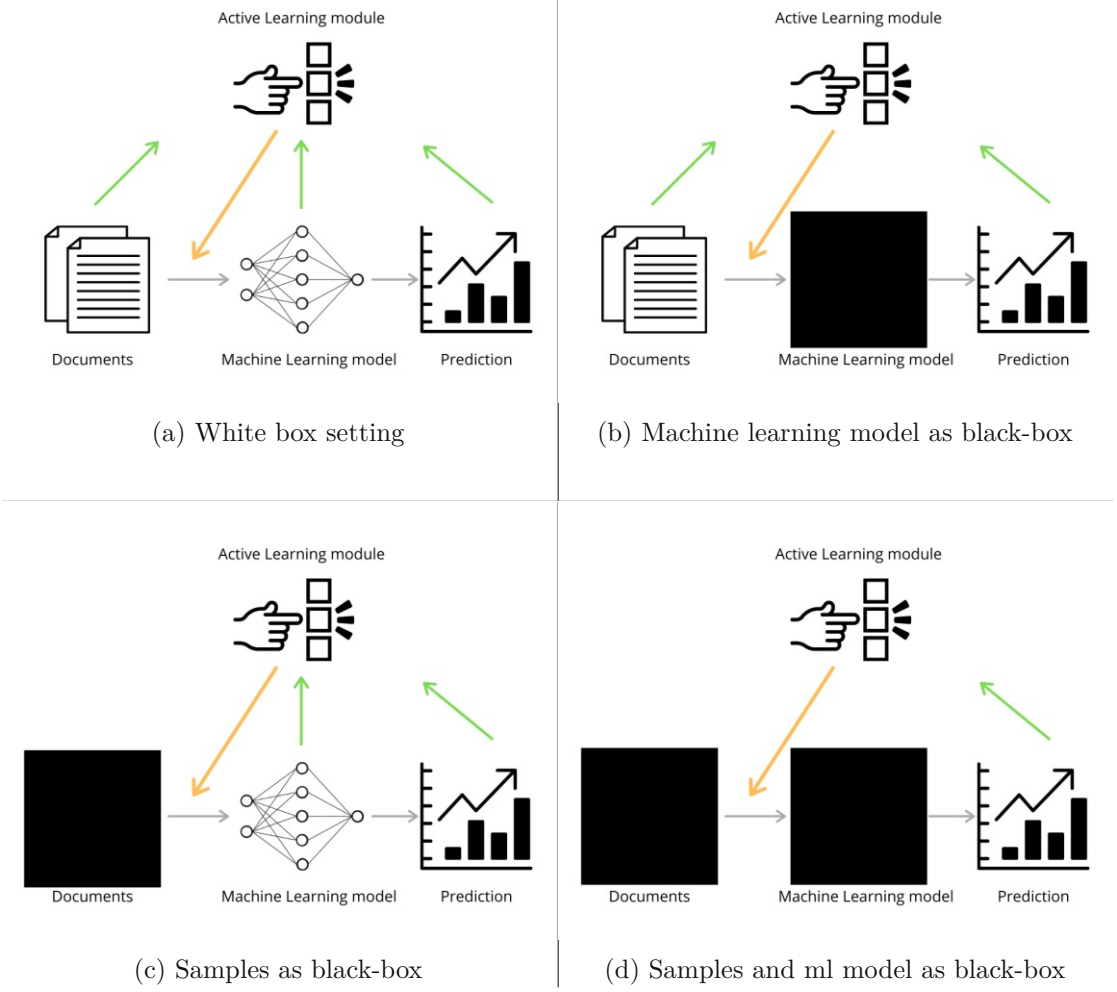
.

so. The human experts are expensive, so in the best case we want them only to label the "set of points such that a model learned over the selected subset is competitive for the remaining data points" [4]. Every case is different in the sense that there are different classes for the documents to fall into. Having only a few training samples available for each class is known as *Few-Shot learning* [5, 6]. Typically few-shot methods use prior knowledge to extend the training data or to constrain the hypothesis space. Prior knowledge can come in explicit form like data from related domains or in implicit form as in pre-trained weights of machine leaning models. In Section 3.2 we will see in detail an example from the latter approach, which we utilize in our case. The idea in [6] is to learn how a model learns, based on only a few training examples. We will see two methods of this type of *meta-learning* in Sections 4.4.1 and 4.4.2.

Double black-box scenario

Another speciality that we face in the project setup is that we only have limited or no access to the base model itself. We do not know the architecture of it and do not know how it was trained. However, we have the possibility to query the model and observe the output, namely the predictions. This set up is well known in other research areas of machine learning, such as *security* of ml-models. In [7] the authors want to mimic a target machine learning model by training an own white box model. They use Active Learning methods to find samples which lead to the best performance of the white box model. The labels are then given by the target model instead of a human annotator. A setting where the model is not accessible is usually referred to as a *black-box setting* [8]. We distinguish between three black-box settings: Either the model, the samples, or

both serve as a black-box. In Figure 2.2 we see a diagram illustrating the three black-box scenarios and the usual white-box scenario where we have access to all parts of the machine learning process. Depending on the scenario some Active Learning methods are applicable and some are not. We will explicitly state in which settings our methods fall into. The situation in the K.Rex project is best described by the fourth scenario in Sub-figure 2.2d. We do not have access to the samples and do not have access to the machine learning model.



(a) White box setting

(b) Machine learning model as black-box

(c) Samples as black-box

(d) Samples and ml model as black-box

Figure 2.2: Four different scenarios for an Active Learning module

With that being said, we continue and categorize the Active Learning methods further by the assumptions they are built on. Mainly three categories can be distinguished [9]:

- uncertainty-based approaches
- diversity-based approaches
- expected model change approaches.

Uncertainty

Uncertainty-based approaches try to select the samples that the model is most uncertain about. Uncertainty is defined with respect to the confidence probabilities a machine learning model outputs. Therefore, methods which fall into this category are suitable for a double-black-box scenario as in our case. A notable example of a uncertainty-based approach is the *query-by-committee (QBC)* algorithm [10, 2]. Here a committee of models $\mathcal{C} = \{M_1, M_2, ..., M_n\}$ is trained on the data available. After that, the classes of the unlabeled samples are predicted. The sample that the models most disagree on is chosen as the one which needs to be labeled. Observation showed that uncertainty-based approaches do often select samples that are highly correlated [11].

Diversity

Samples which are highly correlated are unfavorable for machine learning algorithms. For that, more recent methods try to overcome this problem with diversity-based approaches. Methods which fall into this category try to select a set of data points, which is diverse enough to represent the whole distribution of the unlabeled samples. They are built on the assumption that a diverse training set leads to a high performance of a machine learning model. Just like uncertainty the notation of diversity has to be defined and differs from method to method. A notable example of diversity in the area of image classification is the so called *contextual diversity*, which defines diversity via the spatial neighborhood of objects. The higher the number of potential misleading objects in a set of images is, the higher the contextual diversity for that batch [11].
Any form of uncertainty or diversity can either be obtained via individual predictions [2, 7] or over the whole prediction space [12]. The authors of [12] for example try to split the version space with ever new training sample in two equal parts. They make use of the support vectors employed in support-vector machines.

Relation to statistics

By interpreting the predictions for samples as discrete probability distributions we are able to exploit a range of information theoretical measures. The most famous information measure is probably *Shannon entropy* [13], which we will see in section 4.1.4. The *Akaike information criterion* [14] and *Bayesian information criterion* [15] are measures employed in model selection. They measure a relative information loss between two models and

can therefore be used to select the model with the lowest information loss out of a finite selection of models. Since Active Learning does not select models but rather training samples we do not make use of those criteria.
In order to compare probability distributions we can measure their "distance" to each other. For that, we calculate their divergence [16], which is a function mapping both probability distributions onto the positive real numbers. The only case for the function to be zero is when both probability distributions are the same. Every function which fulfils these two criteria is considered a (statistical) divergence. Notable distance measures between probability distributions are *Jensen–Shannon divergence* [17] , which is sometimes referred to as the information radius, and the *Kullback–Leibler divergence* [18]. We will see in Section 4.2 and 4.3 how we can use such distance measures to obtain a diverse set of samples. Choosing a subset of elements out of a super set is generally known as sampling in statistics [19]. The challenge here is to draw unbiased samples from a pool of samples. This is usually achieved by drawing random samples. Another popular approach is *Thompson Sampling* [20] which samples actions from a distribution of expected rewards. Thompson sampling is usually used to solve the multi-armed bandit problem [21] which we will introduce in the next point. Active Learning itself is sometimes described as *Optimal experimental design* which is for example used for designing experiments to uncover biological networks [22]. The most used criteria in optimal experiment design depend on the so called information matrix which is the inverse of the variance matrix of the models parameters [23]. Different minimization or maximizing efforts on the information matrix lead to different optimalities. For example, maximizing the determinant of the information matrix leads to a "D-optimal" design where the D stands for determinant.

Expected model change

Novel approaches to Active Learning fall into the expected model change category. The intuition behind methods from this category is to predict a model change based on previous model changes. This kind of problem solving is quite a natural approach in the area of machine learning.
A metaphoric description of the problem we would like to solve is the *multi-armed bandit*. In a multi-armed bandit setting [24] every sample is considered a lever of a gambling machine called multi-armed bandit. Each lever (or arm) leads to a different reward. As an example one might think of maximizing the click-through rate for a website. The click-through rate depends on the content shown. While the different contents correspond to the different arms on the slot machine the click rate of a user of the website corresponds to the reward. Which content is shown to the user is learned through the behavior of other users and/or prior visits of the website. In other words the content which we see on our favourite apps is far from being random.
In the context of Active Learning, the different levers correspond to the different samples [25]. The reward is usually the expected change in model performance. It is important to note that the reward is only revealed for the chosen lever, similar to the active learning scenario as well. Only samples which are annotated can be used to (re-)train a model. Based on the revealed reward or model change future samples are chosen. In addition

to the history of revealed rewards other information can be considered when making a decision. Situations where this is possible are usually best described with *Contextual Multi-Armed bandits* [26]. The context is "a set of covariates of fixed dimensionality" [26]. Depending on the scenario, the context can be the samples, the representation of the samples inside the machine learning model (feature space) or just the confidence probabilities, or any combination of those three things [25]. Additionally, one can also exploit external data sources which brings us back to the topic of few shot learning [5]. The most recent approaches focus on the decision whether and when to ask a human expert [27, 6, 28] or how to efficiently teach a learner [29]. Mozannar and Sontag [27] propose a framework where a rejector has to make the decision whether to ask a human expert or a machine learning classifier. The rejector itself is a predictor. In order for the rejector to learn the right decision the costs for the different decisions and their consequences are implemented via the loss function. In [28] a predictor predicts if a a sample needs a human label or not. The input which the predictor receives in this case is two fold consisting of a syntax and semantic vector.
Most methods from the expected model change category employ machine learning models for their purpose. Especially *reinforcement learning* [30] can be used to learn a good policy for decision making. Reinforcement learning is the third machine learning paradigm next to supervised and unsupervised learning. In this work, we touch on all three paradigms. We deal in a supervised fashion with labeled data and in a unsupervised way with unlabeled data which we would like to categorize. Reinforcement learning usually deals with an agent navigating in an environment. The agent has been appointed with solving a certain task. When successful the agent is given a reward. The agent itself is a machine learning model which tries to maximize its reward. Algorithms from the area of Contextual Multi-Armed bandit problems are sometimes referred to as associative reinforcement learning as well [26]. We will see methods based on multi-armed bandits and reinforcement learning in Sections 4.4.1 and 4.4.2.

Human centred approaches

More human centred approaches are found in the area of *Explainable AI* and its connections to active learning [31]. Instead of focusing on the pure information gain a model achieves when trained on one batch of samples over the other, human centred approaches focus on the annotator and the work she has to do. A relatively simple and straight forward approach to reduce the annotation effort is the following: The model presents the predictions to the annotator and annotations are only made as corrections if necessary [32]. So called *coactive leaning* is employed in the document annotator tool used in the K.Rex project as well. A core observation is that "users [of machine learning systems] [...] value transparency beyond performance" [31]. Transparency as achieved by explaining black-box models can either come as *local* or *global* explanations. While global explanations reflect the reasoning process of the whole model, local explanations explain decisions for a single sample. Annotators are often experts in their own field but not necessarily in machine learning. For that annotators might prefer local over global

explanations [31]. Local explanations can come as textual or visual artefacts, which highlight the importance of certain features for the model's decision. A notable algorithm which works like that is the LIME algorithm [33]. Besides highlighting important parts of the input for the model's decision LIME is model agnostic which means that it is perfectly applicable if the model comes as a black-box.

CHAPTER 3

Theoretical Foundations

In this chapter we lay out the theoretical foundations of our tasks and algorithms we use.

3.1 Machine learning task

We formulate the machine learning task in mathematical terms [34, 35]. Let $D = \{(x_i, y_i)\}$ be a labelled data set, where x_i is a feature vector, $y_i \in 1, ..., K$ a class label and $i = 1, ..., N$ the number of feature vector label pairs. Feature vectors are the symbolic representation of samples. We use both words interchangeably.
We have a classification model M, which is trained on training data and makes predictions $y_M(\cdot)$ in the form of giving confidence probabilities [21]:

$$p_M(y = k \mid x) \text{ for each class } k = 1, ..., K. \tag{3.1}$$

Typically one chooses

$$y_M(x) = \underset{k}{\operatorname{argmax}}\, p_M(y = k \mid x) \tag{3.2}$$

as the predicted class, when using model M to predict the class of a sample.
The overall performance of the classification model can be measured with different metrics like Recall, Precision and F1 score, which are all calculated from the confusion matrix $A = (a_{i,j})_{i,j=0..K}$. An entry $a_{i,j}$ in the matrix is the number of samples, which belong to class i and are classified as class j. As an evaluation criterion for our task, we use multi-class accuracy obtained via

$$\text{multi-class accuracy} = \frac{\sum_{i=1}^{K} a_{i,i}}{\sum_{i,j=1}^{K} a_{i,j}}, \tag{3.3}$$

and call this measure *accuracy on classes* or just *accuracy*.

3.2 Transfer learning

Since we use transfer learning [36, 37] for our experiments, we define what we mean by this. For this, we specify our model M, which is a neural network in our case. An important decision when setting up a neural network is the type of layer(s) it is made of. In fact, the type of layers is name-giving to the overall network. Apart from the types of layers, a neural network has several hyper-parameters describing its architecture. These hyper-parameters are related to the number of neurons, number of layers and their connection. Since we will fix these-hyper parameters, we do not define them formally, but just state them when needed. In addition to hyper-parameters, a neural network has parameters that are optimized in the training process. These parameters are typically the weights between the neurons and biases. We will refer to the set of all weights and biases as θ, which gives us a parameterized model M_θ. Usually one starts with randomly initialized weights, which are then changed during training.
The intuition behind transfer learning is that already trained models learn faster than fresh models. In detail, transfer learning works as follows: We start with a model M_{θ_0}, which was trained on training data D_0, which does not have to be from the same domain as the data we use to solve our task. We adapt the architecture of model M to our task respectively data and obtain model M', which is similar to M. The new model is initialized with the parameters θ_0 as far as possible, since the architecture is no longer exactly the same. To be more precise, a large subset $\theta_0' \subseteq \theta_0$ plus randomly chosen weights are used to initialise the new model. We obtain M_{θ_1}, where $\theta_1 = \theta_0' \cup \theta_{\text{random}}$. The new model M_{θ_1} is then trained on new training data D_1.
In practice, one usually uses a state of the art model, which was set up by experts and trained on a lot of data. Adaption to the task specific data is usually made just for the last layer, which means that the last layer is replaced with a layer, which has as many neurons as classes one wants to predict. For training on new data, all weights but the ones for the last few layers are frozen. This way, one can retrain huge state-of-the-art models with low computational power.

3.3 Active learning task

For the active learning task [2], we are given an unlabeled data set $D = \{(x_i)\}$ of size n for which labels can be obtained from human annotators. The human annotators can label a fixed number k of samples $D' \subset D$. The labels $\{(y_k)\}$ together with the respective samples are then used for training a model M. We want the annotators to label the subset that leads to the best performance of model M. The hyper-parameters of the model are not changed. In fact, we do not even have access to them in our project set up. We can observe the output of the model, but not its internal workings. The output of a neural network model are the values of the last layer. We refer to this output also as *annotation vector*. When using a softmax activation function in the last layer, the annotation vector has only non negative values which add up to one. Hence we can interpret the annotation vector as the confidence probabilities in Equation 3.1.

CHAPTER 4

Methods

In this chapter we present and discuss various approaches for Active Learning. Every approach has a different intuition behind it, which we will explain.

4.1 Score calculations

We start off with score calculations. By score calculations we mean that we calculate a score for every sample individually. The n top scoring samples are chosen, annotated and used for training. The scores are calculated from the annotation vectors, i.e. the output of the last layer of the neural network model. The output of the last layer are the probabilities that give the confidence of membership of a class i.e. the confidence probabilities from equation 3.1.

4.1.1 Random selection

Random sampling is the trivial approach, where we select a random subset of all samples. It serves as a baseline when answering the question if Active Learning can help to reduce the number of required training samples.

4.1.2 Least confident

The idea of *least confident* [2, 7] is that we chose the sample x for which the models confidence probability is the lowest:

$$f_{LC}(x) = 1 - p_M(\hat{y} \mid x), \tag{4.1}$$

where $\hat{y} = \text{argmax}_y\, p_M(y \mid x)$. The intuition behind it is that the model needs to be trained on the samples for which it is most uncertain.

4.1.3 Margin sampling

While least confident only considers the confidence probability of the least confident class, *margin sampling* [2, 7] considers the confidence probabilities of the most confident and second to most confident class. If these two probabilities are very close, we want to use this sample for retraining, since there is a high chance of confusing these two classes. The formula is as follows:

$$f_{MS}(x) = 1 - (p_M(\hat{y_1} \mid x) - p_M(\hat{y_2} \mid x)), \tag{4.2}$$

where $\hat{y_1}$ and $\hat{y_2}$ are the first and second most confident classes.

4.1.4 Entropy

While margin sampling considers two probabilities, we discard a lot of information, especially when we deal with many classes. An approach that considers the confidence probabilities of all classes is *entropy* [13, 2, 7]. The formula for calculating entropy of a prediction vector is

$$f_E(x) = -\sum_i p_M(y_i \mid x) \log p_M(y_i \mid x). \tag{4.3}$$

The more similar the class predictions are, the higher the score and vice versa. The entropy for equally distributed confident probabilities is one and zero for a prediction which is 100% confidence for one class.

The next two methods are our contribution and are not mentioned in the literature as far as we know.

4.1.5 Mutual info score

By interpreting the output of the model as a probability distribution, we can use (statistical) divergences to measure distances between two predictions. One of them is the so called *Kullback-Leiber divergence* [18], which is defined as follows:

$$KL(P, Q) = \sum_{k=1}^{K} P(k) \log \frac{P(k)}{Q(k)}. \tag{4.4}$$

While Kullback-Leiber divergence and mutual information are closely related, they are not the same. Even though we use Equation 4.4 for our method, we call the method *mutual info score*. We define the mutual information score as follows: We obtain the uniform distribution of equally confident class predictions and then calculate the Kullback-Leiber divergence with the real prediction probability distribution. In mathematical terms the score is defined as

$$f_{MI}(x) = -KL(P(x), Q), \tag{4.5}$$

where P corresponds to the probability distribution of the predictions of sample x and Q to the uniform one. In Q every class is predicted with probability $\frac{1}{K}$ with K being the

number of classes. The idea behind this method is to measure how similar the predictions are to the useless prediction of equally predicted classes. The more similar the predictions are to the useless prediction, the higher the score of the corresponding sample. That is why we have a minus sign in Equation 4.5.

4.1.6 Difference uniform score

The idea behind the *difference uniform score* is the same as for the mutual info score method. We want a score, which indicates how similar the prediction is to a useless or uniform random prediction. Here, we just take the pairwise difference between each prediction and the useless prediction, and sum them up:

$$f_{DU}(x) = -\sum_i \left| p_M(y_i \mid x) - \frac{1}{K} \right|, \tag{4.6}$$

where K is the number of classes. Once again we negate the sum, such that predictions which are very close to the useless prediction score higher.

4.2 Subset selection

While score calculations are straightforward and computationally inexpensive, they have one big drawback. They are calculated individually for each sample, which means that they do not consider other samples. The information and relation between samples is lost when viewing the samples independently from each other. To overcome this problem, we reformulate our task: We do not want to find the samples, but the optimal subset that leads to the biggest improvement of performance.

4.2.1 Equal pseudo classes

Based on the idea that each class should be represented by an equal number of samples in the subset, we make use of the notion of a pseudo class [11] and introduce the *equal pseudo classes* method. A pseudo class y_M is the class, which the model predicted, so:

$$y_M(x) = \underset{k}{\operatorname{argmax}}\, p_M(y = k \mid x), \tag{4.7}$$

where $p_M(y = k \mid x)$ is the prediction vector of the sample x. Regardless if the prediction is correct or if $y_M(x_i) \neq y_i$, we use the pseudo classes to produce a subset which is balanced out with respect to its pseudo classes. For that, we determine the smallest pseudo class, which is the smallest number of samples belonging to the same pseudo class and take the same number of samples from all other pseudo classes in our subset. In case we did not reach our desired size yet, we add random samples. In case we exceeded our desired subset size, we only take as many samples as we need, but an equal number from each pseudo class. We randomly discard superfluous samples in this case.
This method builds on a single and very strong assumption, namely that classes are of

equal size. Since this assumption does not hold in most real scenarios and the pseudo-classes do not coincide with the real classes, we discard this method and do not use it in our experiments.

4.2.2 Vector norm

This method is our contribution and the idea behind it is to choose the subset of fixed size, for which the predictions are most far away from each other. To measure a distance between two predictions we use the L2 norm:

$$\|p_m(x_i), p_m(x_j)\| = \sum_{k=0}^{K} \sqrt{|p_m(x_i)_k - p_m(x_j)_k|^2}. \tag{4.8}$$

We fill up our set with samples x_i and x_j, for which the norm is greater then all other pairwise norms, until we reach our desired set size. The intuition behind this is that we include samples from various different classes in our subset. Since we do not know the classes a priori, we have to rely on the predictions of the model. This approach might work well when the model makes good predictions already.

4.2.3 Structural similarity for diversity

Next we want a subset of samples which is diverse. The intuition is that the more unsimilar the samples are, the more the model is going to learn about the concept of classes. If we want the model to learn a class "invoice" for example, we do not want to have invoices from only one company in our training samples, but rather invoices from different companies and in different formats. To achieve this, we do not only consider predictions, but also samples itself. Since we implicitly already defined diversity as the inverse of similarity, we need a similarity measure for our samples, which are pictures. We choose the so called *structural similarity* [38] as one of the most famous measures. Structural similarity or the structural similarity index is calculated over windows of the images. The formula for calculating the structural similarity index is a product composed of luminance, contrast and structure of the image. With a suitable choice of parameters the structural similarity index has the following form:

$$SSIM(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2\mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}, \tag{4.9}$$

where

- μ_x is the mean of x,
- μ_y is the mean of y,
- σ_x^2 is the variance of x,
- σ_y^2 is the variance of y,

- σ_{xy} is the covariance of x and y,
- C_1 and C_2 are variables to stabilize the fraction in case of small denominators.

Since the computation for structural similarity between two images is expensive, we do not compute the values for every two samples, like we did it for the metric span in Section 4.2.2. We rather want to have diversity in the different classes. For that we consider all samples that belong to the same predicted class or pseudo class and calculate the structural similarities for these. We then fill up our set with samples x_i and x_j, which have the lowest structural similarity between each other. We further try to balance out the classes by taking an equal number of samples from each pseudo class. If we run out of samples, we add randomly chosen samples.
We note that this method belongs to a setting where the internals of the machine learning model are unknown but the samples are accessible. Since this is not the case in the K.Rex project, we did not use this method in our experiments.

4.2.4 Contextual diversity

The crucial observation of [11] is, that "[a models] misclassification is not simply attributed to the objects from the true class, but also to other classes that may appear in the object's spatial neighborhood." The idea is to find a set of samples which is diverse with respect to the spatial context responsible for misclassification. In our black box setting, we do not have access to the weights describing spatial context, but only to the confidence probabilities. Therefore we adapt the proposed approach from [11] to our case.
The authors assume a model which outputs predictions for regions r of images as $p_r(y = k \mid x) = p_r$ for each class $k = 1, ..., K$. $\mathcal{I}^k \subseteq D$ is the set of samples, where for each sample at least one region is classified in class k and $D = \{(x_i)\}$ is the unlabeled data set. Within a single sample $x \in \mathcal{I}^k$ the set of regions that are classified as class k is denoted by $\mathcal{R}_x^k$. The collection of all regions with pseudo class k is $\mathcal{R}^k = \bigcup_{x \in \mathcal{I}^k} \mathcal{R}_x^k$, which is assumed non-empty for a large number of samples. The class-specific confusion is defined as follows:

$$P_D^k := \frac{1}{|\mathcal{I}^k|} \sum_{x \in \mathcal{I}^k} \left[\frac{\sum_{r \in \mathcal{R}_x^k} \omega(p_r) p_r}{\sum_{r \in \mathcal{R}_x^k} \omega(p_r)} \right], \tag{4.10}$$

where $\omega(p_r) = -\sum_k p_r(k) \log p_r(k) + \epsilon$ and $\epsilon > 0$, the Shannon entropy (Equation 4.3) plus a small constant. Equation 4.10 calculates the class-specific confusion for the whole data set. To compute the mixture for a single sample x the Equation is reduced to:

$$P_x^k = \frac{\sum_{r \in \mathcal{R}_x^k} \omega(p_r) p_r}{\sum_{r \in \mathcal{R}_x^k} \omega(p_r)}, \tag{4.11}$$

where p_r depends on x. In the case where only overall predictions are available, as opposed to predictions on regions of samples, formula 4.11 is equal to the prediction vector $p(x) = p_M(y = k \mid x)$ itself.

For two samples x_1 and x_2 we can now calculate the pairwise contextual diversity as:

$$d(x_1, x_2) = \frac{1}{2} KL(P^k_{x_1}, P^k_{x_2}) + \frac{1}{2} KL(P^k_{x_2}, P^k_{x_1}), \tag{4.12}$$

where KL is the KL-divergence from Equation 4.4. Note that the pairwise contextual diversity is only defined for samples which have the same pseudo class. This is due to the fact that we derived it from the class specific confusion 4.10. For samples which do not belong to the same pseudo-class we define their pairwise contextual diversity as zero. So in our case the distance measure boils down to:

$$d(x_1, x_2) = \begin{cases} \frac{1}{2}(KL(p(x_1), p(x_2)) + KL(p(x_2), p(x_1))), & \text{if } x_1 \text{ and } x_2 \text{ same p.c.} \\ 0, & \text{otherwise,} \end{cases} \tag{4.13}$$

which is the symmetric KL-divergence of the predictions.

4.2.5 Cosine distance

A well known similarity measure used in *information retrieval* is the cosine similarity [39]. Let v and u be two non-zero vectors, then the cosine similarity is the cosine of the angle θ between the two vectors:

$$\text{similarity}_{\cos}(v, u) = \cos(\theta). \tag{4.14}$$

With the dot-product and the L2 norm we obtain the cosine of the angle θ from the two vectors as

$$\cos(\theta) = \frac{v \cdot u}{\|v\|\|u\|}. \tag{4.15}$$

The cosine similarity has the nice property, that it is 1 for when v and u are the same vector and 0 for when they are orthogonal to each other. Since we interpret distance as the opposite of similarity, we define the cosine distance as

$$d_{\cos}(v, u) = 1 - \frac{v \cdot u}{\|v\|\|u\|}. \tag{4.16}$$

The two vectors u and v are the class predictions $p_m(x_i)$ and $p_m(x_j)$ in our case.

4.3 Distance Matrices

The problem of selecting a subset of size k from a super set of size n in order to minimize some norm is NP-complete [40]. To avoid solving this problem, we utilize the fact that Equations 4.8, 4.16 and 4.13 are symmetric distance measures between samples (i.e. their predictions). Symmetric means that for a distance d the equation

$$d(x_i, x_j) = d(x_j, x_i) \text{ for all } i, j \in \{1, ..., N\} \tag{4.17}$$

holds. By calculating the distances between all samples, we obtain a symmetric distance matrix of size number of samples:

$$D(\{(x_i)\}) = \begin{pmatrix} 0 & d_{1,2} & d_{1,3} & ... & d_{1,n} \\ d_{2,1} & 0 & d_{2,3} & ... & d_{2,n} \\ d_{3,1} & d_{3,2} & 0 & ... & d_{3,n} \\ ... & ... & ... & ... & ... \\ d_{n,1} & d_{n,2} & d_{n,3} & ... & 0 \end{pmatrix}. \tag{4.18}$$

The score we attribute to one sample is the accumulated distance, which this particular sample has to all others. For example the score for the sample with id 2 would be the sum of all yellow marked entries in the following distance matrix:

$$D(\{(x_i)\}) = \begin{pmatrix} 0 & d_{1,2} & d_{1,3} & ... & d_{1,n} \\ d_{2,1} & 0 & d_{2,3} & ... & d_{2,n} \\ d_{3,1} & d_{3,2} & 0 & ... & d_{3,n} \\ ... & ... & ... & ... & ... \\ d_{n,1} & d_{n,2} & d_{n,3} & ... & 0 \end{pmatrix}. \tag{4.19}$$

By this means, we obtain scores for every sample. If we want a subset of a given size n, we can sort all samples with respect to their scores and take the top n ones.
We utilize the effectiveness of using a distance matrix, which lies in $\mathcal{O}(n^2)$ (with n being the number of samples) over a subset selection. Based on the three distance measures introduced in Sections 4.2.2, 4.2.5 and 4.2.4 we define three methods, which are

- *Vector Norm method* with L2 norm (Equation 4.8) as distance measure,
- *Diversity Score method* with contextual diversity (Equation 4.13) as distance measure and
- *Cosine distance method* with cosine distance (Equation 4.16) as distance measure.

To calculate the distance matrix we use Algorithm 1. Note that we start the second for-loop from the subsequent prediction vector of the current prediction vector. This is due to the properties of the distance functions: The symmetry gives us the possibility to ignore the lower triangular matrix and the property of two vectors having zero distance saves us this calculation step as well.

4.4 Adaptive methods

Until now, we used static methods to calculate the importance of a sample for retraining. In this section, we explore adaptive methods. The general idea of the following methods is to learn *how* the classifier learns. For this meta learning task, the methods use their own simpler machine learning algorithm which we call *oracle* or *agent* in this context.

Algorithm 1 Distance matrix method

Require: Ordered list of prediction vectors $[p_M(x_k)]_{k=1,...,N}$ of samples made by model M, symmetric distance measure d, empty score list $s = [0, ..., 0]$ of length N
for p_M^i in $[p_M(x_k)]_{k=1,...,N}$ **do**
  for p_M^j in $[p_M(x_k)]_{k=i+1,...,N}$ **do**
    $d_{current} \leftarrow d(p_M^i, p_M^j)$
    $s[i] \leftarrow s[i] + d_{current}$
    $s[j] \leftarrow s[j] + d_{current}$
  end for
end for
return samples ordered by score list s

4.4.1 Contextual Multi-Armed bandit

In general, a Contextual Multi-Armed bandit describes the problem of choosing an action from a possible set of actions, based on a given context in order to maximize the payoff for all chosen actions [24]. The payoff for individual actions is not known before choosing the action and depends on the context. After choosing an action, the payoff is revealed. It is easy to see how this problem adapts to our use case: Here, the context is the samples we want to classify, the class predictions or both. The actions that can be chosen correspond to retraining the classifier with a chosen sample. The payoff is the improvement of the classifier, when trained on a sample. Since there is no known method to know a priori how the classifier will improve, we can only reveal the payoff to the Multi-Armed bandit oracle after retraining with a chosen sample.
The oracle is a machine learning regressor which is trained on the context and payoffs. In more detail, the regressor gets the context (either samples or/and predictions of samples) as input and the corresponding improvements as target values. Choosing an action is realized by predicting the improvements of all not yet annotated samples and then taking the one with the highest expected improvement. The action consists of annotating the sample and retraining the classifier with said sample. The improvement is measured and (together with the sample) provided as feedback to the oracle. The idea is that with more and more context/improvements pairs the oracle learns to predict the improvement of a potential training sample more accurately. We note that instead of choosing individual samples and using them for retraining the classifier, the oracle chooses batches (top n samples) in practice.
In the beginning of the process, the predictions of the oracle are not very good, since the oracle was not yet trained or only trained on few context/improvements pairs. To overcome this issue we chose a random action if the expected payoff is smaller than a defined threshold, i.e. we take a random sample for retraining, if the expected improvement for the top sample is smaller than the threshold. The threshold is decreased with every iteration, since the set of samples to chose from is getting smaller as well [26].
If we want individual scores assigned to samples, we predict the expected improvement

of all training samples and use this value as the score.

4.4.2 Reinforcement learning

Reinforcement learning [30] usually deals with an agent, which has to chose actions $A \in \mathcal{A}$ in an environment in order to maximize some numerical reward $R \in \mathbb{R}$. The environment consists of states $S \in \mathcal{S}$. The revelation of the reward is immediate and the choice of an action influences the environment. Indexing states, actions and rewards by time-steps $t = 0, 1, 2, ...$ we obtain a sequence

$$S_0, A_0, R_0, S_1, A_1, R_1, S_2, A_2, R_2, ..., \tag{4.20}$$

which the agent goes through.
The reward is based on a policy π, which is defined beforehand. The policy depends on the action and the environment. Since every action influences the environment, the same actions can lead to different rewards when executed in different states of the environment. This is extremely important when dealing with long term rewards, which might be received after a long series of actions with low or even negative rewards. In order for the agent to explore long term rewards usually a so called discount factor is used which reduces early rewards.
A finite sequence (Equation 4.20) of consecutive states, action and rewards is called an episode, which is fixed by a maximal number of time-steps or by reaching a desired goal, i.e. a desired state of the environment. For every action in the episode the reward is received and reinforced, such that the agent learns a mapping from actions to rewards. In the first episodes we do not expect the agent to make good decisions. We rather want the agent to explore the environment in the beginning. For that we let the agent choose a random action with a certain probability, which gets smaller after every episode.

Reinforcement learning for training sample selection

In the context of our active learning task we refer to the agent as *oracle*. The oracle chooses an action A from a set of actions $\mathcal{A}$ of size two:

$$A \in \mathcal{A}. \tag{4.21}$$

The set of actions $\mathcal{A}$ consists of: the label should be predicted by the machine learning model or the label should be given by a human annotator. While we can expect the human annotator to give the correct label y, the label predicted by the machine learning model y_M might not be the correct one. The environment which the oracle perceives is the images with the prediction of every image or just the predictions. In the case of asking the human annotator for the correct label the reward R_{rec} is negative and small. When asking the machine learning model for the label, the reward R_{cor} is positive and high if the machine learning model predicts correctly. When asking the machine learning

model for the label and the prediction is wrong, the reward R_{inc} is negative and high. Summarized the reward R is the following:

$$R = \begin{cases} R_{rec}, & \text{if a label is requested} \\ R_{cor}, & \text{if ml model predicts and } y_M = y \\ R_{inc}, & \text{if ml model predicts and } y_M \neq y. \end{cases}$$

The oracle itself is a regressor, which makes two continuous predictions when presented an instance of the environment. The first one is the expected reward when asking a human annotator, the second one is the expected reward when asking the machine learning model. The action which the oracle chooses, is evaluated by taking the option with the highest expected reward. To be more precise, the oracle receives a sample and the prediction of the sample (or just the prediction in the case of black box training sample scenario) and predicts the expected reward for both actions. The action is chosen and the real reward for this action is obtained. The decision is reinforced by retraining the regressor with the sample (and/or prediction of the sample) and the reward pairs for both action options. The reward pair consists of the predicted reward for the option which was not chosen and the real reward for the option which was chosen. In our use case the actions of the oracle do not influence the rewards of future actions. While we use the predictions of rewards to chose an action, which leads to retraining the machine learning model and in the best case to better predictions of said model, the action if we chose to ask the human annotator or the machine learning model do not change the reward of a future action directly. In other words, asking the human annotator or the machine learning model for the label of a sample does not influence the reward for the same decision for another sample in the future. For that we omit the discount factor described earlier in Section 4.4.2.

Human method and model method

The oracle from Section 4.4.2 predicts two things: The expected reward when asking the human annotator and the expected reward when using the machine learning model for labeling a sample. In a stream based approach we would take the option with the highest reward as we described before. When dealing with a pool of unlabeled samples we have more options to make use of the oracles learnings. Namely we can assign two scores to each sample in the pool. The scores correspond to the expected rewards for asking either the human or the model for the prediction. For choosing an Active Learning batch we order the samples with respect to their expected reward for asking the model. The top-n samples make up the Active Learning batch. We call this approach the *Oracle Model method*. When ordering the samples with respect to the expected reward when asking the human, we take the top-n samples as well, but start with the sample with the lowest score. This is due to the fact, that we want to minimize the reward for asking the human annotator and maximize the reward for asking the machine learning model. We call the latter approach the *Oracle Human method*.

4.4.3 Regressor

Both methods described in Section 4.4.1 and 4.4.2 are built around an oracle, which is a regressor [35]. The regressor in the Contextual Multi-Armed bandit context learns and predicts the improvement of the machine learning classifier directly. The regressor in Reinforcement learning learns and predicts rewards for the decision human annotator vs. machine learning model based on a defined policy. The input for the regressor can be either the sample plus the prediction of the sample by the machine learning model or just the prediction of the machine learning model of the sample. In the first case, we call the oracle algorithm a *dual-regressor* and in the second case a *mono-regressor*. The mono-regressor only receives a probability vector of length number of classes. For that case we can use any trainable regressor like Random Forests or (deep) neural nets. The dual-regressor receives two-dimensional images plus the probability vector of length number of classes. For the first input channel we introduce convolutional layers, which we describe in more detail in Section 5.2.1. These layers (together with layers of other types) will break the images down to a vector. The vector is concatenated with the second input channel, which is the probability vector. The result of this operation is fed into a neural net with the desired output dimension. Mono-regressors are used when we are dealing with a black box scenario regarding the training data itself. Dual-regressors can be used when only the machine learning model is unknown to us. The multi-channel dual regressor we used for our experiments can be seen in Figure 4.1. The left channel is used to process the images in convolutional, pooling and drop-out layers, while the right (short) channel just serves as an input for the prediction vectors. The final convolutional layer is flattend and concatenated with the input from the right channel. This high dimensional vector is then send through a series of dense and drop-out layers until it reaches the final output layer with two neurons, corresponding to the expected reward for the two possible decisions.

4.4.4 Epsilon Greedy training

As machine learning algorithms the mono and dual regressors from Section 4.4.3 require training. To train the regressor and make use of reinforcement learning we use algorithm 2, ϵ-greedy for oracle training. The algorithm describes the sequence in Equation 4.20 in more detail. The states S_t from the sequence correspond to the prediction of the samples by the model M in case of a mono oracle and the prediction of the samples by the model M plus the samples themselves in case of an dual oracle. We highlighted the difference of input in Algorithm 2 in red. The actions A_t correspond to the decision the oracle makes, while the revealed reward R_t is based on the reward policy π. The oracle is initialized randomly at the beginning so we do not expect it to make good decisions which maximise the reward. Due to that we let the decision be random with a certain probability ϵ. The name giving ϵ parameter is initialized with 0.5 at the beginning, which means that the oracle's decision is random with 50% probability. With every episode we decrease ϵ, such that the probability for a random decision gets smaller. This is realized by the decay factor, a number below 1, which gets multiplied with ϵ in each episode. The probabilities

for the random decision itself do not change and are equal for both options. In case the decision is not random it is based on the predicted rewards of the oracle. The option with the higher expected reward is taken. The shift from random decisions to decisions made by the oracle is known as *exploration to exploitation* [30].
After the reward is obtained a sample we retrain the oracle with a reward tuple as target. The reward tuple consists of the received reward for the option which was taken for the respective sample and the predicted reward for the option which was not taken. We denote the latter reward with $r_{\neg \text{ decision}}$.
In order to reveal the reward to the oracle we need annotated samples. Since we chose a pool based approach we do not annotate the samples which the oracle predicts the highest reward for but a set of samples. Consequently we use Algorithm 2 every time we obtain new annotated documents in order to retrain the oracle. In our experiment set up we use ϵ-greedy after the initial fine tuning of the machine learning model and after every round of Active Learning. The two Active Learning methods oracle model and oracle human method from Section 4.4.2 always use the most recently trained regressor as an oracle.

4.5 Summary

The methods we deploy can be categorized into three families: Score calculations, distance matrices and adaptive methods. The score calculating methods calculate scores for individual samples. The n highest scoring samples are chosen by the methods. The distance matrices methods calculate the accumulated distance of each sample to all other samples. The n most furthermost samples are chosen by the methods. As far as we are informed a usage of a distance matrix in such a way for Active Learning was not yet mentioned in the literature. The adaptive methods use a regressor oracle to predict a score. The score is either the expected improvement or an expected reward. Based on that score the top n samples are chosen by the methods.

Algorithm 2 ϵ-greedy for oracle training

Require: model M, oracle O, reward policy $\pi = (R_{rec}, R_{cor}, R_{inc})$, samples
  reward $R \leftarrow 0$
  $\epsilon \leftarrow 0.5$
  decay factor $\leftarrow 0.999$
  for all episodes **do**
    $\epsilon \leftarrow \epsilon \cdot$ decay factor
    for all samples **do**
      prediction vector $p_M \leftarrow M$ predicts classes for sample
      if random number between 0 and 1 $< \epsilon$ **then**
        decision $\leftarrow$ random
      else
        $(r_{\text{model}}, r_{\text{human}}) \leftarrow O$ predicts rewards for both options on base of (p_M, sample)
        decision $\leftarrow$ option with higher reward
      end if
      if decision is to go with M's prediction **then**
        if M's prediction is correct **then**
          $R \leftarrow R + R_{cor}$
        else
          $R \leftarrow R - R_{inc}$
        end if
      else
        $R \leftarrow R - R_{rec}$
      end if
      $(r_{\neg \text{ decision}}, R) \leftarrow$ obtain reward tuple
      $O \leftarrow$ retrain O on (p_M, sample) as input and $(r_{\neg \text{ decision}}, R)$ as target
    end for
  end for
  return trained oracle O

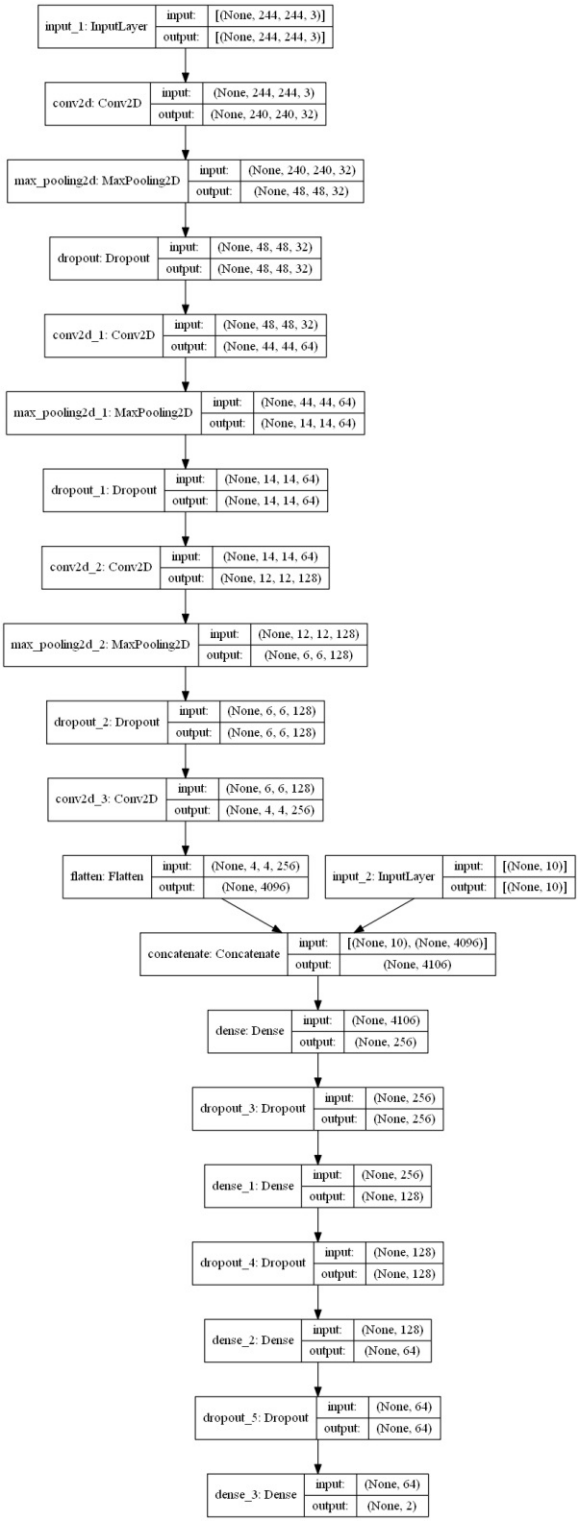


Figure 4.1: A multi-channel regressor with Convolutional and Dense layers

CHAPTER 5

Experiment Set Up

We carry out experiments to test and compare the different methods described in Chapter 4. The experiments are done on two different data sets, both consisting of documents. The first data set is a publicly available data set, the second data set consists of non-public documents, obtained during a legal investigation. While we are able to access the documents in the first case, this is not possible for the second case. The (machine learning) task we want to solve is the same for both data sets: To classify documents into classes in a supervised fashion. The machine learning algorithms we use to solve this task are different for both data sets. We describe in detail how the experiments are set up in the following sections.

5.1 Data sets

To our knowledge there exists only one publicly available document data set suitable for machine learning, which is the Tobacco data set [41]. The real data we use in our experiment was obtained during a legal investigation and annotated by a team of legal experts, developers and the author during a designated annotation session.

5.1.1 Tobacco

The Tobacco data set [41] consists of "[...] approximately seven million documents (roughly 40 million scanned pages in TIFF format) [which] became public through legal proceedings against five US tobacco companies and two tobacco industry research institutes" and "were scanned by the tobacco industry using diverse technologies" [41]. For our experiments we use a subset of the original Tobacco data set, which is called Tobacco3482 [42]. With 1,7 GB it is significantly smaller than the original data set, which takes up 1.5 TB on a hard drive [41]. The small size allows to train and retrain different classifiers and to try out various methods on it, all relatively fast even with low

computational resources.
Tobacco3482 consists of 3.492 black and white jpg images. The images are classified into ten classes, which are: 'ADVE (advertisement)', 'Email', 'Form', 'Letter', 'Memo', 'News', 'Note', 'Report', 'Resume', 'Scientific'. In Figure 5.1 we see three example images from different classes.

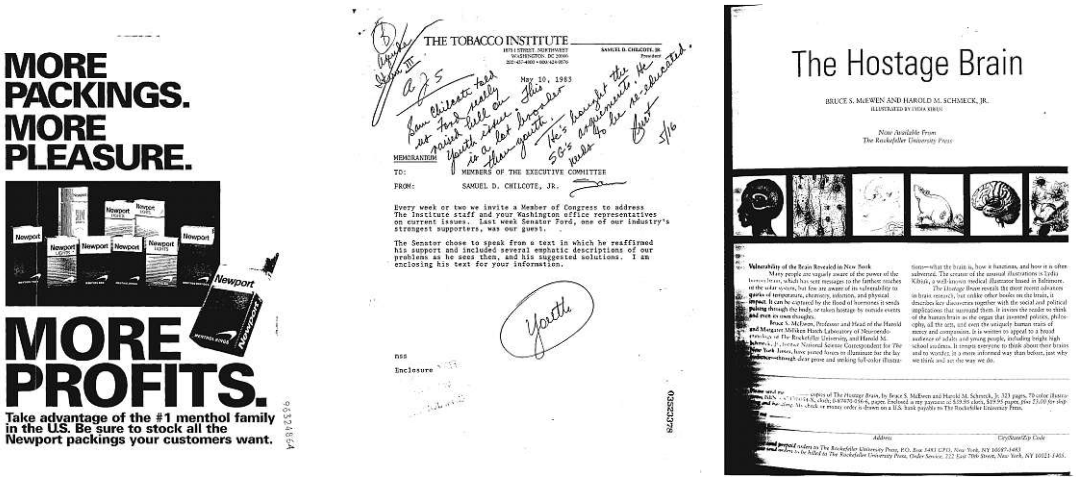


Figure 5.1: Three images from the classes 'ADVE', 'Memo' and 'Scientific' (from left to right)

For the experiment we split the data set into three parts. The first part is used for initial training of the machine learning classifier, the second part is used for testing the performance of the classifier and the third part is used to carry out the active learning experiments described later in Section 5.3. The distribution of classes is balanced throughout all three parts. The splitting into the three parts is random, which is realized by shuffling the data set before splitting. We state the split for each experiment individually.

5.1.2 Real Data

In the scope of the K.Rex project a data set was created. The data set consists of documents of a legal person. Since the documents are coming from a real use-case , we refer to this data set as real data opposed to the Tobacco data, which is mostly used in academia. The documents themselves are not known to us and only accessible by the machine learning algorithm, which we describe in Section 5.2.3. We know that the documents consist of digital documents and scanned paper documents which also include handwritten text. All documents are OCRed, which is important for textual machine learning models. We assume that the documents were collected randomly and we know that the documents are unlabeled. Since a supervised approach was chosen in the K.Rex project, we require labeled documents for the machine learning algorithms to be able to learn. While the idea is, that Active Learning will reduce the labeling effort, the methods

and the machine learning model itself were not ready when it came to labeling the data. The labeling was done in so called annotation sessions. The annotation sessions took several days and included experts from the legal domain, law enforcement and developers. The goal of the annotation sessions was to label the documents into one of 34 predefined classes. The total number of documents is 4057. We note that there was a second task achieved in the annotation session, which was the labeling of individual segments of the documents. Recognising segments and classifying them into predefined categories is outside of the scope of this thesis.

5.2 Machine Learning Algorithms

In order to classify the documents, we use state of the art machine learning algorithms. For the public data presented in Section 5.1.1 we use a so called Convolutional Neural Network, which only considers the visual aspects of the images. For the real data presented in Section 5.1.2 a multi-modal approach is used, which considers visual and textual information, when solving the task of classification. The multi-modal approach was developed by project partner RSA FG.

5.2.1 Convolutional Neural Network

When dealing with image data the number of features is extensive. Typically an image comes as a two dimensional array with three colour channels. Simply connecting all those values (pixels) in a fully connected manner throughout several layers, would lead to an extensive amount of parameters. To overcome this problem Convolutional Neural Networks (CNNs) exploit the fact that "images have a strong 2D local structure: variables (pixels) that are spatially nearby are highly correlated" [43]. For the general architecture of CNNs three ideas are utilized:

- Local receptive fields
- Shared weights
- Sub-sampling.

Local receptive fields: Each neuron in a layer receives a convoluted input from a set of units from the previous layer, which are located in a small neighbourhood. Typically these neighbourhoods are defined as quadratic sets of pixels.
Shared weights: Visual artefacts, like edges, corners and end-points usually occur on several locations on the image. For that, a convolutional layer consists of different planes made up by units. The units in a plane share all the same weights and their output is called a feature map. That means, that one plane extracts the same features all over the image. The planes themselves have different weights, which leads to the extraction of different features. All in all a convolutional layer extracts different features of every part of the image.

Sub-sampling: After detecting a feature, the exact location of the feature is not important anymore. We even want to avoid the network to learn the exact location of features, because the location may vary for different instances from the same class. In order for the network to not learn the exact position we reduce the spatial resolution of the feature maps. In order to achieve this blurring we could average over sets of pixels and exchange the set with the obtained average value. Another method would be exchanging the set with the maximal value.
For general Convolutional Neural Networks, the first two ideas are manifested in the name giving convolutional layers. So called pooling layers implement sub-sampling. The classification happens in fully connected layers which come after the convolutional and pooling layers. An example of an architecture is shown in Figure 5.2.

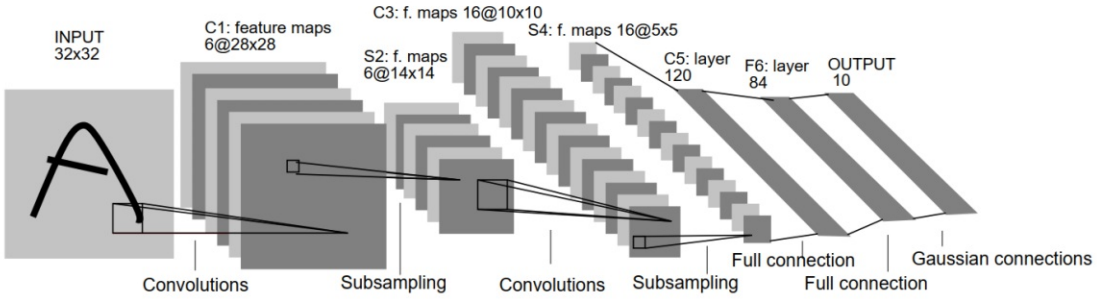


Figure 5.2: An example architecture of a CNN from [43] for digit recognition

Inception Network

For our experiments we use a pre-trained inception network [44]. An inception network is an extension of an ordinary Convolutional Neural Network described in Section 5.2.1. Instead of convolutional layers an inception network has inception layers. Inception layers contain several convolutions of different size and typically a pooling block [45]. An example of such an inception layer is visualized in Figure 5.3. The intuition behind this architecture is "visual information should be processed at various scales and then aggregated so that the next stage can abstract features from different scales simultaneously" [45]. The convolutional blocks can be trained in parallel. The outputs of the building blocks are then concatenated and passed to the next layer.
It is easy to see that the output of a naive inception layer has a very high dimension, resulting from concatenating several building blocks. In order to avoid a model with too many parameters, dimension reduction is applied. The dimension reduction is realized by 1×1 convolutions. By using a lower number of filters then in the layer before the 1×1 convolutions reduce the number of filters while keeping the same spatial dimensions. A schematic visualisation of this process can be seen in Figure 5.4.

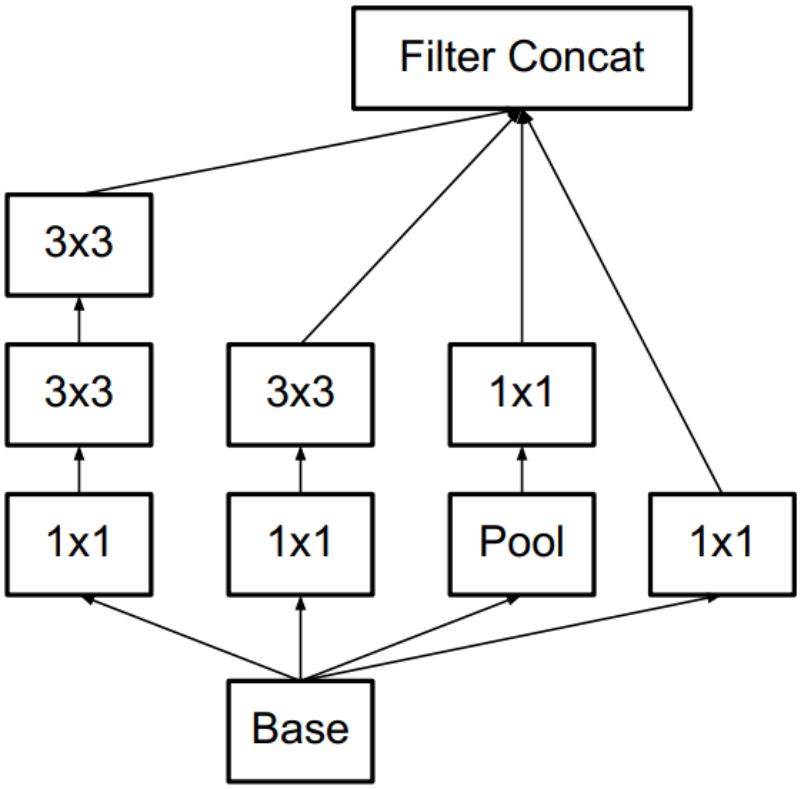


Figure 5.3: An inception module from [44], where different convolutions process information in parallel

The final inception model from [45] is set up with two pure convolutional layers and nine inception layers. For our purpose we use transfer learning as described in Section 3.2 to adapt the inception network to our data set and task. For that we exchange the top layers of the original model with two fully connected relu activated layers with 64 neurons each. For the last layer we use a dense layer with ten neurons representing the ten classes from our data set described in Section 5.1.1. Here we use softmax activation, such that the outputs are interpretable as prediction probabilities, which sum up to one and are non-negative. We end up with 21,938,730 parameters, which are optimized during training. We note that all but the new top layers have pre-trained weights. The new top layers are initialized with random weights. For that we freeze all pre-trained weights and only train the top layers on the new data. After that we unfreeze the remaining weights and train the whole network.

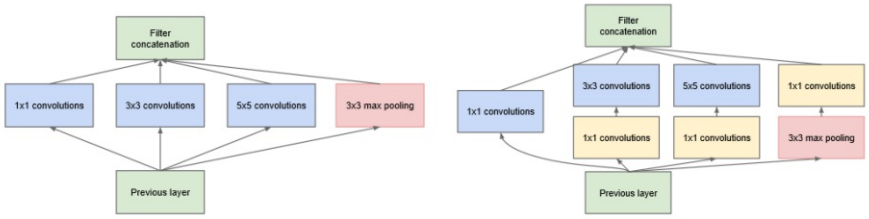


Figure 5.4: Left: Naive Inception module, right: Inception module with dimension reductions from [45]

5.2.2 Model training

The presented models are all machine learning models which require training. The training of a machine learning model aims to optimize the parameters of the model in such a manner that the error of the predictions is minimized. Since we are dealing exclusively with neural network models here, the parameters of the models are the weights between the neurons. We use weights and model parameters interchangeably.

Loss function

In order to measure the error of a prediction we need a *loss function*. The loss function we use here is cross-entropy:

$$L_{CE} = -\sum_{i=1}^{k} y_i \log(p_i), \tag{5.1}$$

where y_i is the true label, p_i the confidence probabilities from Equation 3.1, for the i-th class. A prediction with a high probability for the correct class leads to a small loss while a bad prediction, where the probability is small for the correct class leads to a high loss. The overall loss function

$$\mathcal{L}_{CE}(w) = \frac{1}{n} \sum_{i=1}^{n} L_{CE}(w)_i \tag{5.2}$$

is calculated from the losses of all n training samples. Note that in Equation 5.1 the confidence probabilities depend on the weights w, hence the overall loss function in Equation 5.2 depends on the models parameters as well. Our objective is to minimize the overall loss function for all training samples, so

$$\min_{w} \mathcal{L}_{CE}(w). \tag{5.3}$$

Optimization

In order to minimize the loss function for a model, we optimize the weights of said model. This is done by so called *mini-batch optimization*. While we are using batches of samples for our Active Learning experiments, we divide these batches into smaller, hence mini,

batches which we feed into the network. We then use all samples of these mini-batches to update the weights in order to achieve the objective. The procedure, also known as *Stochastic Gradient Decent (SGD)* [46, 47] in its standard form, is as following:

- calculate the gradients $\nabla \mathcal{L}_{CE}(w)$
- update weights via $w = w - \eta \nabla \mathcal{L}_{CE}(w)$.

This procedure is repeated for a fixed number of rounds, called epochs. The parameter η is the learning rate and regulates the size of "steps" in which the algorithm converges towards the minima. A high learning rate leads a fast convergence of the model, but at the same time might oscillate around the minima. There might even be the case that a training process with a high learning rate diverges. A small learning rate on the other hand leads to a slow convergence of the model, but is safer in terms of not missing a minimum. Typically one might adapt the learning rate during training [48].
Apart from adapting the learning rate, there are several other expansions of the SGD algorithm, which promise even better results [47]. For our training we use the Adam optimizer, which adapts the learning rate η based on the first-order and second-order moments of the gradient [49].

Early Stopping

We noted that training was already converging after a relatively small number of epochs. In order to save computational power we employed so called *early stopping* [50]. The idea of early stopping is straight forward. If there is no change in performance, the training stops. For that, we monitor the performance or the model after each epoch and compare it with the performance from the epoch before. For the experiments based on the Tobacco data set from Section 5.1.1, we chose to monitor the accuracy on the held out test set. Additionally to the performance measure we want to monitor, we set a patience parameter which is the number of epochs the training continues while there is no improvement of performance. In our case we set it to 5 epochs. All in all, the training stops before reaching the maximum number of epochs, when accuracy on the held out test set is not improving for 5 consecutive epochs.

5.2.3 Multi-modal approach

We describe the basic idea behind the K.Rex multi-modal machine learning approach developed by project partner RSA FG, as this framework is used in our experiments. The multi-modal approach aims to classify the documents by two means which are visual and textual. The visual and textual aspects are both captured by individual machine learning models. We refer to these models as the visual and text model. The outputs of both models is combined into a network model which is doing the classification. The output of the network model is a probability vector of size 34 which is the number of classes. We refer to the output also as annotation vector or simply prediction. We call the combined

model Multi-modal model. The Multi-modal model is trained in an end-to-end fashion on batches of documents. The batches are chosen by our Active Learning methods.

5.3 Experiments

We seek to answer the research questions stated in Section 1.2. In order to answer these questions we set up an experiment. The experiment design aims to compare the performance of a machine learning algorithm with different methods for training data selection. The procedure of the experiment is as follows:

1. Initial fine tuning with initial training batch consisting of randomly chosen samples
2. Base performance estimation with hold out test set
3. Selection of retraining batch with Active Learning method
4. Retraining of machine learning classifier with retraining batch
5. Performance estimation with hold out test set
6. Selection of retraining batch with Active Learning method
7. Continue until all samples were used for (re-)training.

The procedure is repeated for different active learning methods. After selecting a new retraining batch there are two possibilities for the retraining. Either we only use the selected batch or we use the selected batch plus all already used samples. Since computational costs for training a classifier are not a limiting issue in the experiments and training data is rare, we favour the second option where we accumulate all samples. The usual procedure to appraise an Active Learning method is to evaluate the Active Learning method against a random selection of samples. We noted that the random selection leads to a really random performance of the models for both data sets. A random performance means that the performance fluctuates too much when comparing several runs with randomly chosen training samples. One solution would be to average over a series of random procedures, but this is too time intensive. So we increase the difficulty level and use the entropy method with the score from Section 4.1.4 as a baseline and evaluate all other methods against it.
For the outcome of the experiment we can expect three cases. Either an Active Learning method leads to a better, a worse or to the same performance of the machine learning classifier. In case of a better performance of an Active Learning method over a baseline selection of training samples, we can answer the research questions. This is done by giving *the delta of samples* which is needed to reach the same performance level when using the worse performing random or entropy selection method. To give a delta we fix a minimum accuracy which we want to achieve with our model. We then just look for the number of training samples needed to achieve that accuracy, for different Active Learning

methods. To give a data set independent value for delta we give the relative difference instead of the absolute value. We answer the first research question in the form: "To achieve an accuracy of x % we need Δ_{rel} % less training samples when using Active Learning method y instead of no Active Learning method (or the entropy method)". For a visualisation of this difference in numbers of training samples needed, see Figure 5.5, where we plotted two training processes.

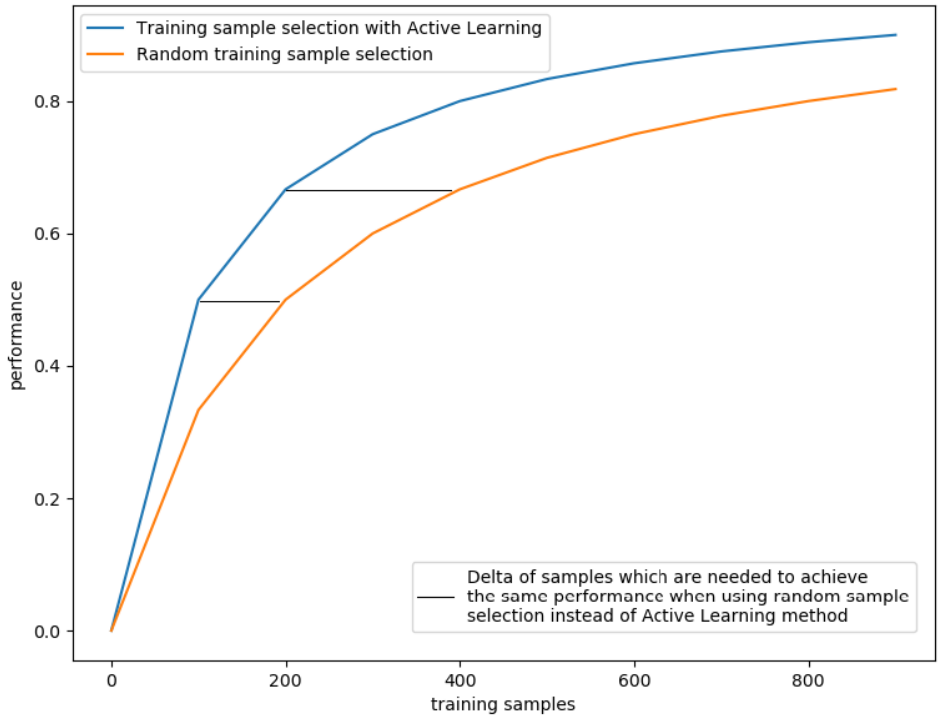


Figure 5.5: Delta of samples which are needed to achieve the same performance when using passive learning over Active Learning

5.3.1 Active learning for Transfer Learning

We do not train our machine learning algorithm from scratch but rather make use of pre-trained weights. This process of transfer learning, as described in Section 3.2 is often used to adapt a classifier from one domain to another. For our experiment based on the public documents for example, we apply a pre-trained Inception Network which was trained on the data used for "The ImageNet Large Scale Visual Recognition Challenge (ILSVRC)" [45]. The ILSVRC data set is composed of images from image hosting services like Flickr [51]. The domain we are interested in is (legal) documents. While we can safely

assume that pictures of documents are part of the ILSVRC data set, we still experience a domain shift here.
It is possible to use active learning for Transfer Learning as well. This is realized by choosing the training data for the new domain with Active Learning methods. If we want to evaluate our Active Learning methods for the transfer learning part of our experiment we simply adapt the split of the data set to 0% for initial fine-tuning. This way, we start with a classifier from a different domain than our target domain and with random initialized last layers. We expect the classifier to learn fast on the new data, especially when the data is selected with Active Learning methods.
We note, that in the case of the real data the initial fine-tuning split is not exactly 0, but 0.05%. This is due to the software architecture in the K.Rex project. In order for the model to be initialized it needs a minimal number of training data. For consistence reasons we chose the same setting for the public data set experiments as well.

5.3.2 Active learning for Fine-Tuning

After adapting a classifier to a different domain using Transfer Learning, the Fine-Tuning phase begins. Fine-Tuning in this context means that we train the classifier on increasing amounts of data. The difference here compared to the situation where we use Active Learning for Transfer Learning is that the initial fine-tuning batch is non empty. We recall that the initial fine-tuning batch consists of randomly chosen samples in this scenario.

5.4 Technologies and Settings

We specify the technologies used to carry out our task and the experiments attached to it. In general, the implementation was done with Python version 3. The machine learning algorithms were set up with the keras/tensorflow frame work. Training and retraining of the CNN model in Section 5.2.1 for the public data set were carried out on a GTX 1080 TI. Training was initialized with 100 epochs and with the early stopping mechanism described in Section 5.2.2.

5.5 Summary

We run two different experiments to evaluate the different Active Learning methods. The first one is done on a public data set with a CNN as a classifier. The second experiment is done on real data gathered in the K.Rex project and a multi-modal approach which was developed by RSA FG. The experiments consist of repeated rounds of choosing training samples based on an Active Leaning method and retraining the classifier with the chosen training samples plus the previous samples. This is done until all samples are used for training. By giving the delta of samples a model needs less to achieve the same accuracy when using an Active Learning method over no Active Learning method, we can answer the first research question. By comparing the performances of the different Active Learning methods we can answer the second research question.

CHAPTER 6

Evaluation

In this chapter we present and discuss the results from the experiments. Before that we define our evaluation method which we then apply to the results. On top of that we answer the research questions asked at the beginning of this thesis.

6.1 Evaluation method

To evaluate the different methods against each other, we introduce a measure of the performance development over the Active Learning rounds. The measure basically measures the relative increase of performance for each Active Learning round and sums them up. Since early rounds are more interesting to us we weight each round such that later rounds contribute less to the measure. We call the measure *relative improvement score* and define it mathematically as follows: We fix a pool of samples and do n rounds of choosing training batches by the Active Learning method of interest and (re-)train a model on this batch. The n batches have sizes $k_1, k_2, ..., k_n$ and $k = \sum_{i=1}^{n} k_i$ is the number of all samples. Let $p(k_i)$ be the performance measure of the model after it was trained on training batch k_i. In our case $p()$ is multi-class accuracy Equation 3.3. The relative improvement score r is then defined as

$$r := \sum_{i=1}^{n} \frac{p(k_i) - p(k_{i-1})}{k_i} \left(\frac{k - (\sum_{j=1}^{i} k_i) + k_1}{k} \right). \tag{6.1}$$

In case the overall number of samples k differs from experiment to experiment, the formula is as follows:

$$r^* = \frac{1}{k} \cdot r. \tag{6.2}$$

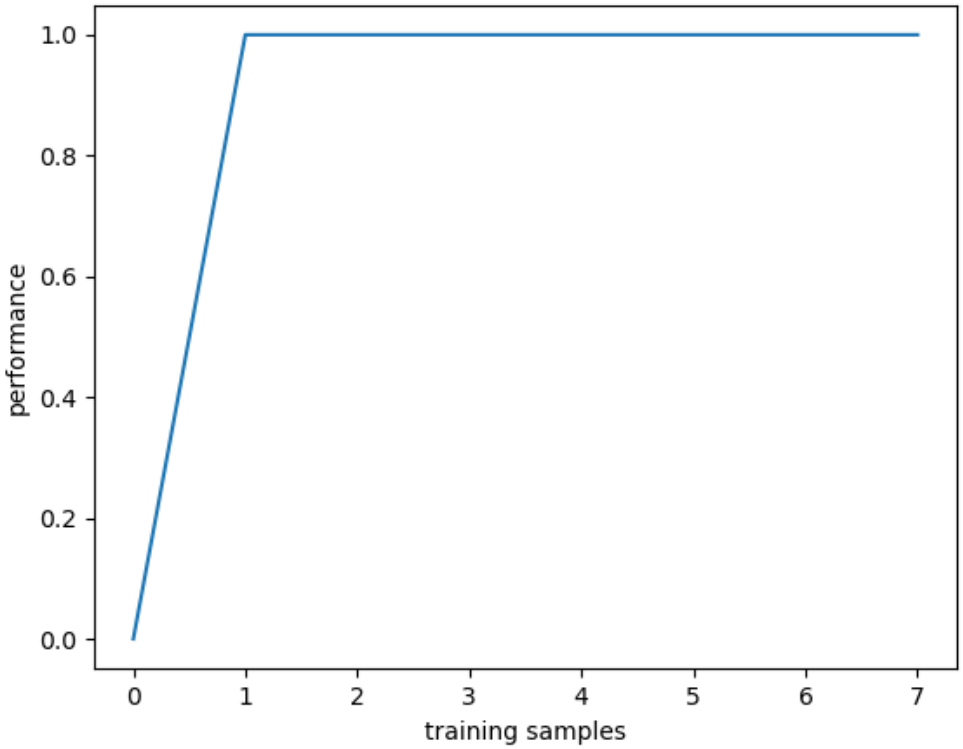


Figure 6.1: A perfect training sample leads to a maximal performance of the model

It often makes sense, that the batches have the same size, so $k_i = k_j$ for all i, j. In this case Equation 6.1 boils down to

$$r = \sum_{i=1}^{n} (p(k_i) - p(k_{i-1})) \left(\frac{n - i + 1}{n} \right). \tag{6.3}$$

The first factor in the sum of Equations 6.1 and 6.3 is the relative change of performance after training with a new batch of samples. The second factor determines the weights of the individual batches. For the first batch, the weight is 1. It decreases linearly for every next batch. The intuition for this score is that a perfect training sample should lead to a score of 1. A perfect training sample is a sample which, when used for training a model, increases the performance from 0 to 1 and every consecutive training step would not decrease the performance. If a model is trained on such a hypothetical sample in the first round the training would look like in Figure 6.1. The relative improvement score of such a training development is 1. In general we favour training curves that increase very rapidly and this at the beginning of the training meaning when trained on the first samples or batches. In Figure 6.2 we see two training curves where we would favour the blue

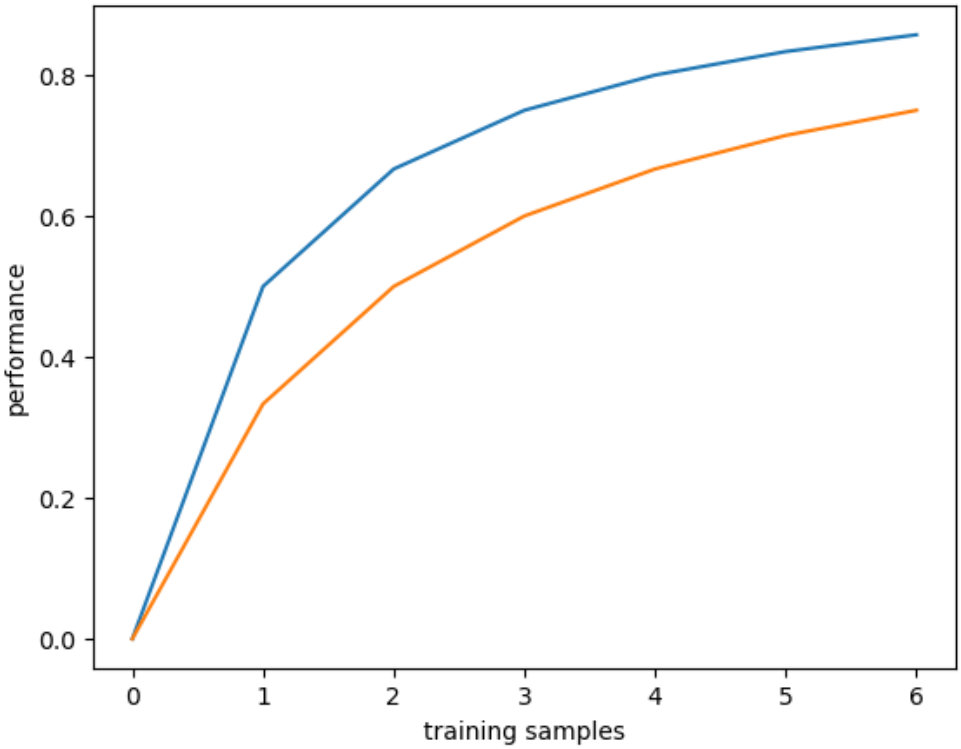


Figure 6.2: Two performance developments

one. Fixing a batch size of one for this example, the relative improvement score is 0.735 for the blue training development and 0.594 for the orange training development. An Active Learning method which chooses the training samples that lead to the performance development with the higher score is evaluated as the better choice.

6.2 Public Data set

The experiments on the Tobacco data set from Section 5.1.1 were carried on with the Inception Network from Section 5.2.1. We fine tuned the network with 5% of the whole data set which were randomly chosen. After that we ran five consecutive rounds of Active Learning, meaning that we chose a batch of samples with a designated Active Learning method and retrained the Inception network on that batch. The initial training and retraining was initialized with 20 epochs and the early stopping mechanism described in Section 5.2.2. For all methods, the chosen batches were of the same size such that we can compare the different methods. The held out test set was made up by 20% of the whole data set. To avoid confusion we link the methods to the sections where we describe them:

- *Entropy*: Score Equation 4.1.4
- *Margin sampling*: Score Equation 4.1.3
- *Least confident*: Score Equation 4.1.2
- *Mutual info uniform*: Score Equation 4.1.5
- *Diff Uniform*: Score Equation 4.1.6
- *Cosine distance method*: Distance matrix Section 4.3 with cosine distance Equation 4.2.5
- *Contextual diversity score*: Distance matrix Section 4.3 with contextual diversity score Equation 4.13 as distance measure
- *Vector norm method*: Distance matrix Section 4.3 with L2 norm Equation 4.8 as distance measure
- *RL oracle model*: Model method from Section 4.4.2 with a mono regressor from Section 4.4.3 as oracle which only sees the predictions made by the Inception model
- *RL oracle human*: Human method from Section 4.4.2 with a mono regressor from Section 4.4.3 as oracle which only sees the predictions made by the Inception model
- *RL dual oracle model*: Model method from Section 4.4.2 with a dual regressor from Section 4.4.3 as oracle which sees the predictions made by the Inception model and the documents themselves
- *RL dual oracle model*: Human method from Section 4.4.2 with a dual regressor from Section 4.4.3 as oracle which sees the predictions made by the Inception model and the documents themselves.

The methods which are not listed here do not get reported. In particular the methods working with a regressor predicting the model's performance change directly as described in the multi-armed bandit settings in Section 4.4.1 had a too low cost-benefit factor to be included: In order to obtain a single training sample for the multi-armed bandit solver we need to retrain the Inception model. The performance of the overall approach did not justify this effort.
We further note that the RL dual oracle model and RL dual oracle human method are not applicable in the K.Rex project since our algorithms do not have access to the samples themselves. Nonetheless we intended to see if we have a benefit if it happened to be that we have access.

6.2.1 Results

We report the results in numbers and as plotted performance graphs. We include the results of the entropy method in every table and graph for a baseline comparison. The original idea of comparing everything against a random choice of samples was dropped due to the experience we made with the real data. In short, the random sampling method produces too noisy results that are not suitable for comparison.
The results of all methods can be seen in Figure 6.3. As we can see the performance of the Inception model differs quite dramatically depending on the method. The absolute accuracy difference between the best and worst performing method is 36% in the first retraining cycle. The overall best performance was achieved after the last Active Learning round with the reinforcement learning method which orders the sample as if we would ask the model for the label instead of the human annotator. The Inception model achieved an accuracy of 79% here. In order to get a better understanding, we report every family of methods on its own.

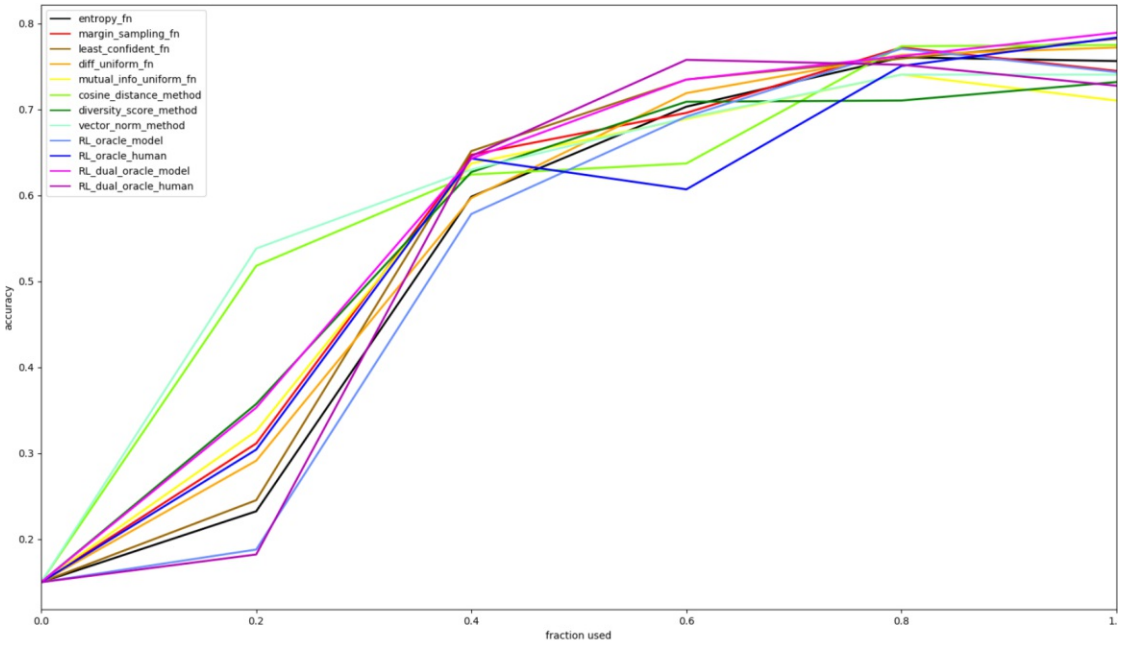


Figure 6.3: Plotted accuracy on held out test set for Inception model (Section 5.2.1) when trained on samples chosen by the respective Active Learning method

Score calculations

The family of score calculating methods calculate a score for every sample individually. As a side note we call attention to that this fact makes these methods suitable for a stream-based scenario. The samples with the highest scores are included in the retraining batches. In Figure 6.4 we see the plotted accuracies of Table 6.1. In the first round the mutual information uniform method scores 10% higher then the entropy method. In the consecutive rounds the performances are getting close to each other with least confident taking the lead for round two and three. Mutual information uniform leads to the worse performance in the last three rounds. If we look at the relative improvement score in Table 6.4 we see that entropy performs worse of all score calculation methods. This comes as a surprise: Margin sampling and least confident only consider respectively two and one entries of the prediction vector while entropy considers all of them. Under the objective criteria of the relative improvement score the best performing score calculation method is least confidence. On the second place we have margin sampling and diff uniform which have nearly the same score.

Fraction used	Entropy	Margin sampling	Least confident	Diff uniform	Mutual info uniform
0.0	0.15	0.15	0.15	0.15	0.15
0.2	0.23	0.31	0.25	0.29	0.33
0.4	0.6	0.65	0.65	0.6	0.64
0.6	0.7	0.7	0.73	0.72	0.69
0.8	0.76	0.77	0.76	0.76	0.74
1.0	0.76	0.74	0.78	0.77	0.71

Table 6.1: Accuracy on held out test set for Inception model (Section 5.2.1) when trained on samples chosen by the respective Active Learning score calculating method

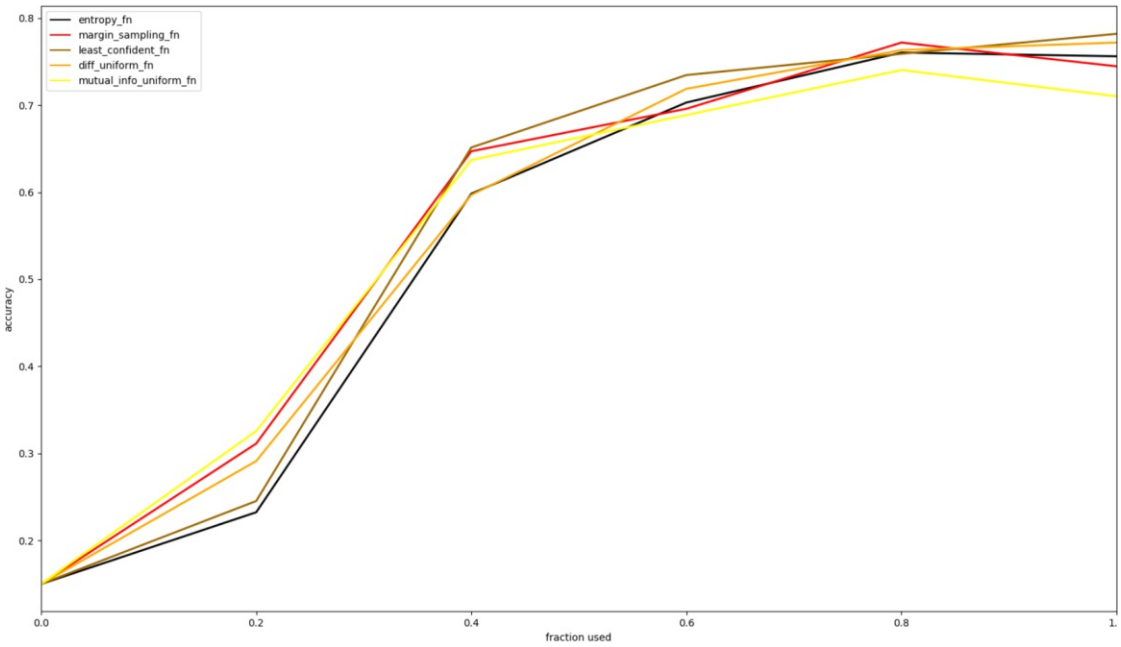


Figure 6.4: Plotted accuracy on held out test set for Inception model (Section 5.2.1) when trained on samples chosen by the respective Active Learning score calculating method

Distance methods

Like we stated before the best performing method in the first round is vector norm. The difference between vector norm and entropy is 31% in the first retraining round as we can see in Table 6.2 and in Figure 6.5. In general the distance based methods perform significantly better than entropy in the first two rounds. The main difference to the score calculating methods is that the distance based methods consider the whole pool of samples for their decision. They chose the samples which have the highest accumulated distance to all other samples with respect to a distance measure. When the pool gets smaller and smaller the advantage of these methods vanishes.

Fraction used	Entropy	Cosine distance	Contextual diversity score	Vector norm
0.0	0.15	0.15	0.15	0.15
0.2	0.23	0.52	0.36	0.54
0.4	0.6	0.62	0.63	0.63
0.6	0.7	0.64	0.71	0.69
0.8	0.76	0.77	0.71	0.74
1.0	0.76	0.77	0.73	0.74

Table 6.2: Accuracy on held out test set for Inception model (Section 5.2.1) when trained on samples chosen by the respective Active Learning distance method

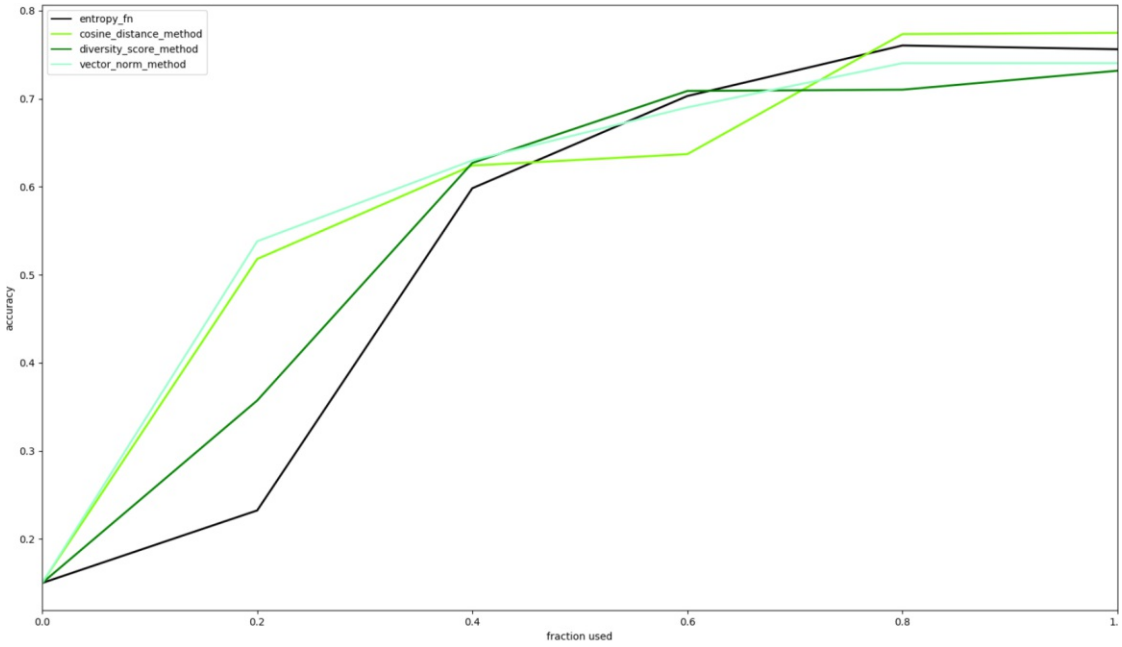


Figure 6.5: Plotted accuracy on held out test set for Inception model (Section 5.2.1) when trained on samples chosen by the respective Active Learning distance method

Reinforcement oracles

The reinforcement oracle methods predict a policy based reward for a decision regarding every sample. They do this based on previous decisions they made when presented with samples. The previous samples the oracles are trained on are exactly the samples we used for transfer learning, so the 5% for initial fine-tuning the Inception model on the

Tobacco data set. After each Active Learning round the oracle gets retrained just as the Inception model based on the exactly the same samples. The difference between the mono and dual oracle is that the dual oracle also gets the sample themselves as an input additionally to the predictions vectors. In Table 6.3 and Figure 6.6 we see the results of the experiment with the oracle methods. In the first retraining round we notice that the RL oracle human method outperforms the entropy method by 7% and the RL dual oracle model method outperforms the entropy method by 12%. The other two methods lead to a 4 and 5% worse accuracy then the entropy method. The two dual oracle methods dominate the performance in round two and three. The mono oracle human method leads to a drop in accuracy in round 3 of 3%. Under the measure of the relative improvement score which we see in Table 6.4, there is no clear trend either. We can not say if the mono or dual methods or the model or human methods are better. We can only conclude that if we decided to go with a dual oracle we should rely on the rewards for asking the model and if we went for a mono approach we should chose the rewards when asking the human annotator.

Fraction used	Entropy	RL oracle model	RL oracle human	RL dual oracle model	RL dual oracle human
0.0	0.15	0.15	0.15	0.15	0.15
0.2	0.23	0.19	0.3	0.35	0.18
0.4	0.6	0.58	0.64	0.64	0.65
0.6	0.7	0.69	0.61	0.73	0.76
0.8	0.76	0.77	0.75	0.76	0.75
1.0	0.76	0.74	0.78	0.79	0.73

Table 6.3: Accuracy on held out test set for Inception model (Section 5.2.1) when trained on samples chosen by the respective Active Learning reinforcement method

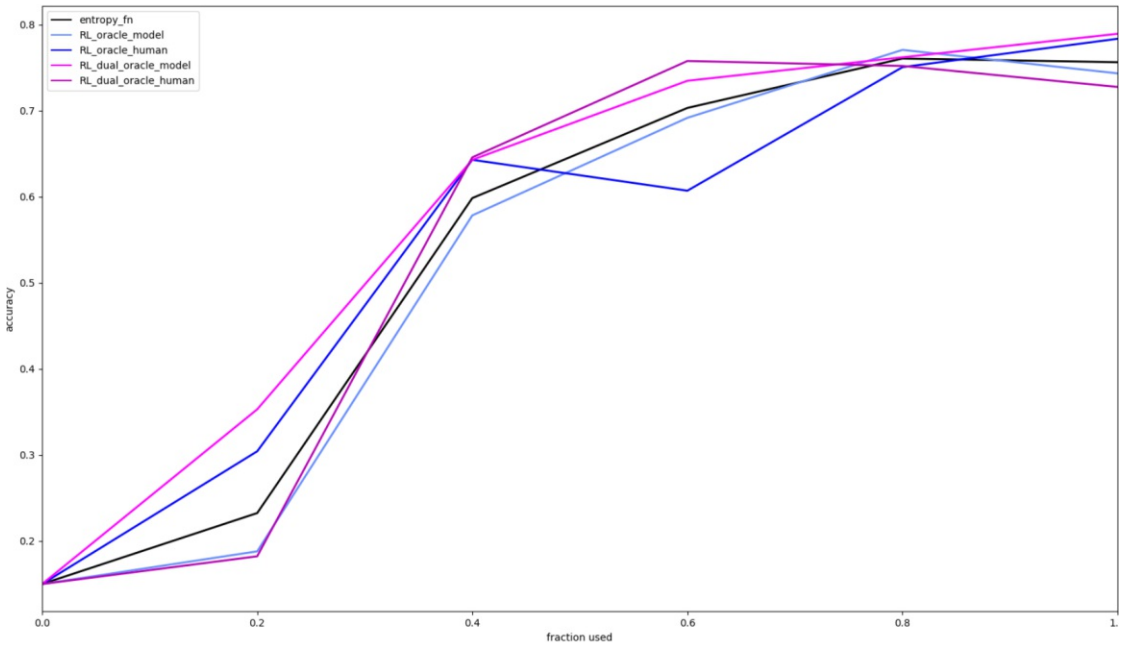


Figure 6.6: Plotted accuracy on held out test set for Inception model (Section 5.2.1) when trained on samples chosen by the respective Active Learning reinforcement method

6.2.2 Discussion

In this section we discuss the results from the experiments on the Tobacco data set with the Inception model as a machine learning classifier and answer the research questions asked in Section 1.2. In the last two rounds the performance more of less stays between 71% and 79% which means that roughly three quarters of the documents are classified correctly. We recall that after the last round all documents were used for training at least once regardless of the method which picked them. Therefore, we focus on the first rounds of retraining the model. Here we have two dominant distance based approaches that increase the performance from 15% base performance to more than 50% accuracy with just 20% of training samples. In comparison entropy achieves 23% accuracy with 20% training samples and 60% accuracy with 40% of training samples. So we can safely conclude that when the objective is to built a classifier which classifies documents correct in 50% of cases we can reduce the annotation effort by half by using the distance matrix method based on either cosine distance or vector norm. If we fix a performance of minimum 75% accuracy that we seek to achieve with our classifier we can give the delta of samples we need for that as well. The entropy baseline method leads to a performance of 75% with 80% of training samples. When using the reinforcement dual oracle human

method we achieve this accuracy with 60% of the training data. Consequently, we can reduce the annotation effort by 25% for this target performance. It is clear that learning methods like the four reinforcement oracle methods increase their performance with more training samples. Since our ultimate task is to reduce the need for training samples this realisation does not help much. We will see in the conclusion Section 7 how we could still leverage on that effect.

Method	Relative improvement score
Cosine distance	0.5336
Vector norm	0.5298
RL dual oracle model	0.5284
Least confident	0.5090
Margin sampling	0.5026
Diff uniform	0.5023
RL oracle human	0.4951
Contextual diversity score	0.4944
Mutual info uniform	0.4853
Entropy	0.4844
RL dual oracle human	0.4820
RL oracle model	0.4691

Table 6.4: Relative improvement scores (Equation 6.3) of all methods on the public data

In order to answer the second research question on which Active Learning performs best, we compare the *relative improvement scores* for all the methods. The relative improvement score was introduced in Section 6.1 and is designed to give methods which lead to an early increase of performance a high score. In Table 6.4 we list all the methods in decreasing order with respect to their relative improvement scores. Under this measure the cosine distance method performs best.

6.3 Real data

We ran three rounds of experiments for the real document data set. The first one compares the score methods described in Section 4.1, the second one compares the two distance metric methods from Section 4.3 and the third one compares the two reinforcement methods from Section 4.4.2. We expected the performance to improve for each round due to the increasing complexity of the methods. For that we included the best performing method of the respective previous round in the second and third round.

6.3.1 Results

For all three rounds we ran 5 active learning cycles as described in Section 5.3. As stated before the entropy methods serves as a baseline for the other methods.

1. Evaluation round (score methods)

In this round the model was fine-tuned on 25% of the data while 20% of the data were held out as the test set. The remaining 55% were split into the 5 active learning batches chosen by the respective method. The results for the first round can be seen in Table 6.5. To compare the methods we plot the performance values against the fraction of samples used. We do this for all the models, the plot can be seen in Figure 6.7.

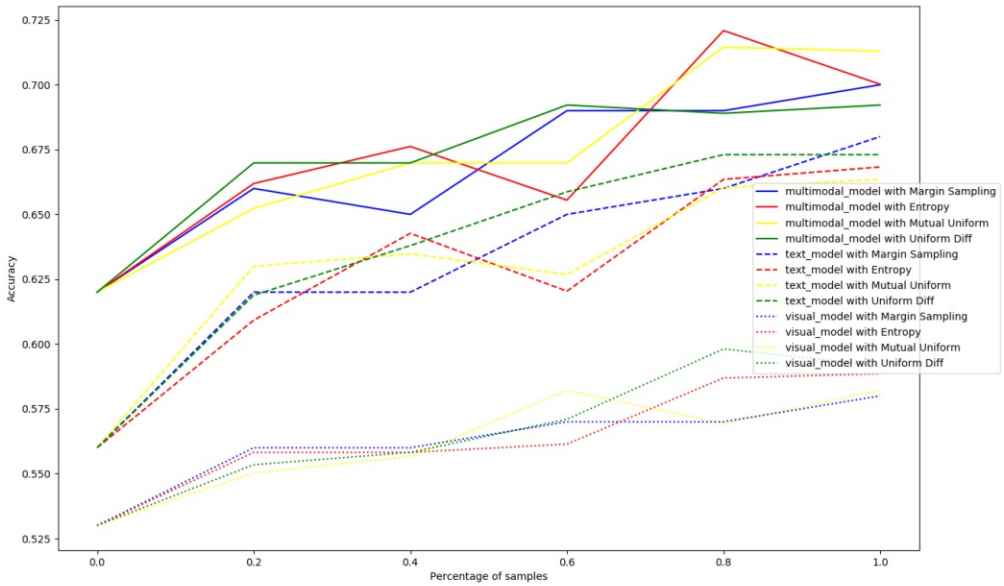


Figure 6.7: Plotted accuracy on held out test set of visual, text and multi-modal mode, when trained on samples chosen by respective Active Learning method

Percentage of samples	Visual model	Text model	Multimodal model
Margin Sampling			
0	0.53	0.56	0.62
20	0.56	0.62	0.66
40	0.56	0.62	0.65
60	0.57	0.65	0.69
80	0.57	0.66	0.69
1	0.58	0.68	0.70
Entropy			
0	0.53	0.56	0.62
20	0.56	0.61	0.66
40	0.56	0.64	0.68
60	0.56	0.62	0.66
80	0.59	0.66	0.72
1	0.59	0.67	0.70
Mutual Uniform			
0	0.53	0.56	0.62
20	0.55	0.63	0.65
40	0.56	0.63	0.67
60	0.58	0.63	0.67
80	0.57	0.66	0.71
1	0.58	0.66	0.71
Uniform Diff			
0	0.53	0.56	0.62
20	0.55	0.62	0.67
40	0.56	0.64	0.67
60	0.57	0.66	0.69
80	0.60	0.67	0.69
1	0.59	0.67	0.69

Table 6.5: Accuracy on held out test set of visual, text and multi-modal mode, when trained on samples chosen by respective Active Learning method

2. Evaluation round (distance methods)

For this round we changed the split of the data set into 5% for initial fine tuning and 10% as a test set. The remaining 85% were used to evaluate the Active Learning methods. Additionally to the two distance metric methods we included Uniform Diff as this method was performing best in the previous round. The results can be seen in Table 6.6. Again we plot the performance values for all models against the fraction of samples used. The plot can be seen in Figure 6.8.

Percentage of samples	Visual model	Text model	Multimodal model
Vector Norm			
0	0.33	0.40	0.33
20	0.41	0.49	0.50
40	0.45	0.52	0.52
60	0.48	0.61	0.60
80	0.55	0.64	0.65
1	0.56	0.63	0.69
Diversity Score			
0	0.33	0.40	0.33
20	0.38	0.51	0.53
40	0.44	0.61	0.61
60	0.50	0.63	0.67
80	0.51	0.65	0.65
1	0.55	0.68	0.71
Uniform Diff			
0	0.33	0.40	0.33
20	0.37	0.55	0.57
40	0.47	0.63	0.63
60	0.47	0.65	0.68
80	0.54	0.66	0.68
1	0.57	0.67	0.70

Table 6.6: Accuracy on held out test set of visual, text and multi-modal mode, when trained on samples chosen by respective Active Learning method

3. Evaluation round (reinforcement methods)

We keep the data set split as before, so 5% for initial fine-tuning, 10% for the test set and 85% for the Active Learning methods and included the best performing method from the previous run again. As Uniform Diff outperformed the two distance metric methods we compared this score method against the human and model method. The results of this last run can be seen in Table 6.7. As before we plot the performance values for all models against the fraction of samples used. The plot can be seen in Figure 6.9.

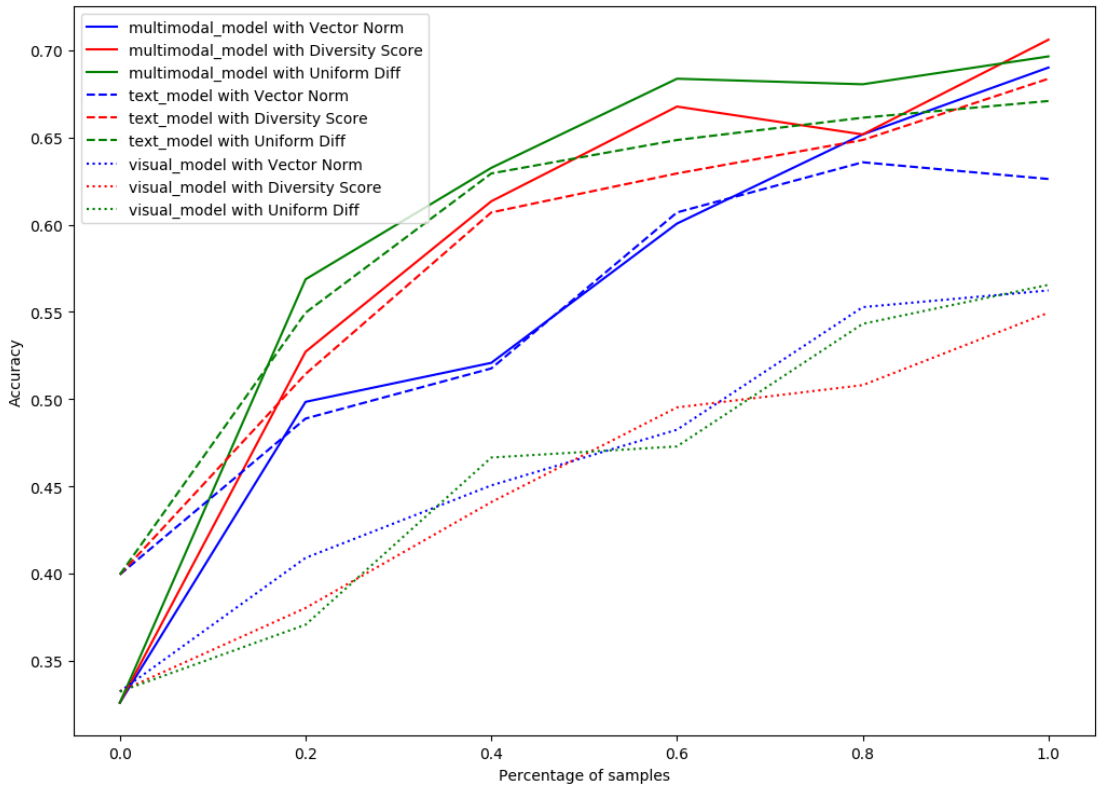


Figure 6.8: Plotted accuracy on held out test set of visual, text and multi-modal mode, when trained on samples chosen by respective Active Learning method

6.3.2 Discussion

Machine learning task

We make a number of observations, which we summarize and discuss. Starting with three main observations which are independent from Active Learning and are general for the machine learning task:

- When using 100% of the samples the different methods do not achieve the same performance
- Training with more data sometimes leads to a drop in performance
- The text model outperforms the visual model in all cases and sometimes even the multi-modal model.

Percentage of samples	Visual model	Text model	Multimodal model
Oracle Human			
0	0.30	0.44	0.31
20	0.35	0.58	0.58
40	0.49	0.53	0.56
60	0.48	0.63	0.66
80	0.50	0.65	0.69
1	0.57	0.70	0.73
Oracle Model			
0	0.30	0.44	0.31
20	0.28	0.53	0.51
40	0.41	0.57	0.56
60	0.49	0.65	0.64
80	0.48	0.65	0.66
1	0.54	0.69	0.70
Uniform Diff			
0	0.33	0.40	0.33
20	0.37	0.55	0.57
40	0.47	0.63	0.63
60	0.47	0.65	0.68
80	0.54	0.66	0.68
1	0.57	0.67	0.70

Table 6.7: Accuracy on held out test set of visual, text and multi-modal mode, when trained on samples chosen by respective Active Learning method

The first point might be confusing since when using all samples one might expect the same performance for the machine learning model. This is given the strong assumption that training samples are independent from each other. Independent from each other means that the ordering in which a model receives the training data does not matter. Regardless of the assumption being true or not the reason for the different model performances with all training data is another one. As describes in Section 5.3 we accumulate the training batches chosen by the Active Learning method. This means that when training with 100% of the training samples, the model was already trained 4 times on the first chosen batch, 3 times on the second batch, etc. These batches differ of course depending on the Active Learning method.
The second observation of a drop in performance can be caused by over-fitting [52] or a distribution shift in the training data. To avoid such behavior of the model early-stopping as described in Section 5.2.2 can be used. It is clear that a model which scores a higher accuracy is preferred over a lower scoring model even when the latter one was trained on more data.
The last point comes as no surprise since we are dealing with text documents. What surprises is that the text model sometimes outperforms the multi-modal model as well

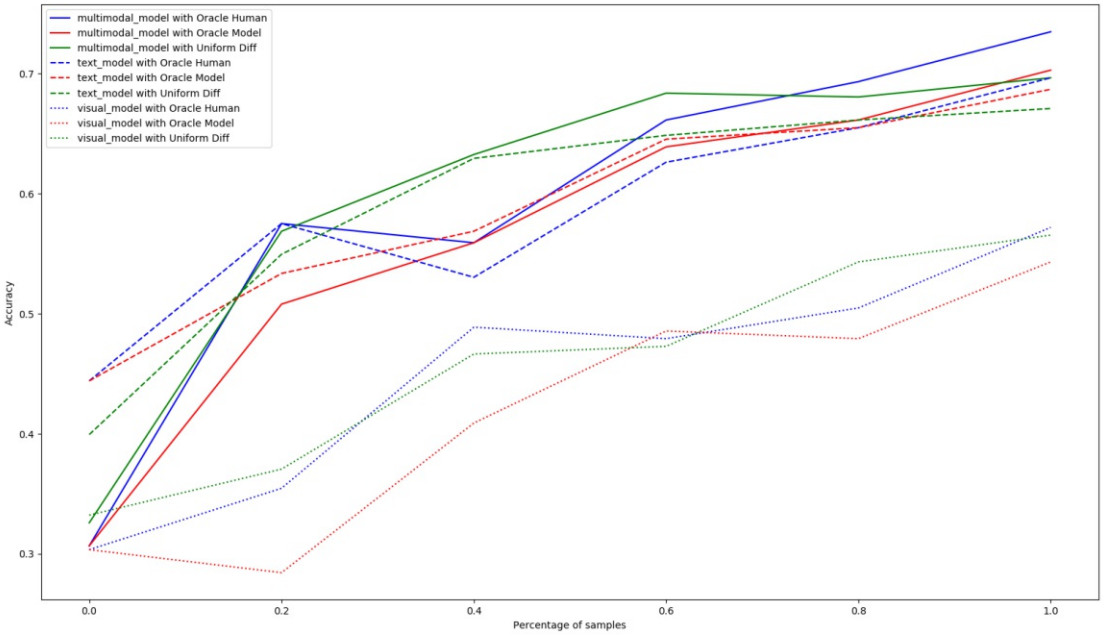


Figure 6.9: Plotted accuracy on held out test set of visual, text and multi-modal mode, when trained on samples chosen by respective Active Learning method

even though the multi-modal model is composed of the text model and the visual model. For example in Figure 6.9 we see the samples chosen by the Oracle Model method lead to a text model outperforming the multi-modal one. Apparently the multi-modal model has not yet learned to trust the predictions of the text-model. The relative low performing visual model for this case could also pull the performance of the composed model. Since we cannot access the models in this case we have to be content with assumptions.

Active Learning methods

Next we move to the discussion of the individual Active Learning methods. As we can see in Figure 6.7 the text model benefits the most from the score methods. Since the score calculations are based on the output of the multi-modal model we can conclude that the multi-modal model prioritises the text model. The two highest overall values are achieved by the entropy method on 80% of training samples with 72% accuracy and by the Oracle Human method with 73% accuracy when using 100% of the training data. Since we are interested in reducing the number of samples we need to annotate, using (and annotating) 80% or even 100% of available samples is not an option we consider. So while these two methods rank highest with respect to highest accuracy achieved overall,

we are interested in the methods which outperform the other methods in the first or second active learning round.
For the score methods these are the Entropy and Uniform Diff methods. They both achieve 66-68% accuracy with the first two chosen batches already. The accuracy for Entropy drops significantly again for the third round. Interestingly the Mutual Uniform method leads to the highest increase for the text model in the first round and to the lowest increase in accuracy for the visual model. The multi-modal model scores the lowest for this method in the first active learning round. The margin sampling method leads to the best performance of the visual model in the first two rounds. The accuracy for the multi-modal model increases significantly in the last three rounds when using margin sampling which is not what we want. All in all Uniform Diff leads to the accuracy development we can leverage on. The highest relative increase of accuracy is in the first round and there is no drop of accuracy when more samples are used for training in the later rounds.
For the distance methods from Section 4.3 we make similar observations as for the score methods. As we can see in Figure 6.8 the Uniform Diff methods dominates the two distance metric methods on nearly all Active Learning rounds. When comparing the two distance method which each other the Diversity Score method outperforms the Vector Norm method. For all three methods the predictions of the textual model are nearly as good (and in one case even better) then the ones of the multi-modal model. With one exception the models do not drop their accuracy in any of the Active Learning rounds.
As we can see in Figure 6.9 the Oracle Human method leads to the highest relative increase in accuracy for the multi-modal model. Unfortunately the accuracy drops again in the next round by 2%. The Oracle Model method does not lead to a drop in accuracy for the multi-modal model, but for the visual model in the first round. Still the Oracle Model only outperforms the Uniform Diff method by merely half a percent. Once again the Uniform Diff method outperforms the other methods especially in the first few rounds.
To make a more quantitative statement on which method is suitable for the task we use the evaluation metric which we introduced in Section 6.1. Since we always chose the same size of batches in each evaluation round 1,2 and 3, we use Equation 6.3 to calculate the relative improvement score. Note, that the first evaluation run was done to experiment in the setting of Fine-Tuning an already trained classifier as described in Section 5.3.2. Evaluation run 2 and 3 where done to experiment in the setting of Active Learning for Transfer Learning where we trained a classifier (almost) from scratch on a new domain as described in Section 5.3.1). For that we present the relative improvement scores in two tables. Table 6.8 is contains the scores for evaluation round 1 and Table 6.9 for evaluation rounds 2 and 3.

	Entropy	Margin Sampling	Mutual Uniform	Uniform Diff
Relative improvement score	0.0629	0.0580	0.0639	0.0626

Table 6.8: Relative improvement scores of evaluation round 1

	Oracle Human	Oracle Model	Uniform Diff	Vector Norm	Diversity Score
Relative improvement score	0.3380	0.3073	0.3265	0.2665	0.3073

Table 6.9: Relative improvement scores of evaluation round 2 & 3

Based on this metric the results look a bit different then before. The best performing score method is Mutual Uniform, which received the highest relative improvement score (highlighted in yellow). Even though this method scores highest only in the last Active Learning round the relative improvement for this method over all Active Learning rounds is the highest. We see that the Entropy and Uniform Diff methods are pretty close in terms of the score they received with Entropy scoring a bit higher.
When comparing the more complex methods which each other the Oracle Human method scores the highest with Uniform Diff on a tight second place. The Oracle Model and Diversity Score receive the same relative improvement score while the Vector Norm method scores the lowest. Considering the complexity of an reinforcement learning oracle, which needs to be trained on a predefined policy and needs training data itself, the Uniform Diff method is still the better choice. This method needs nothing but a prediction vector to calculate its score. This is done independently from all other predictions and the computational costs for the score calculation are negligible.

CHAPTER 7

Conclusion

In order to classify documents with machine learning algorithms like Convolutional Neural Networks we require training data. The data must come as annotated samples. When gathering new data the samples are often not annotated meaning they miss a class label. Since annotation is done by humans and costly we employed Active Learning methods to minimize the number of samples we need to annotate. To evaluate the usage of Active Leaning methods for training sample selection we compared the difference of samples which are needed to achieve the same performance when using no Active Learning methods over using Active Learning methods. In order to compare the different Active Learning method we used a newly introduced relative improvement score. The relative improvement score is defined such that it gives a high value to Active Learning method that improve the performance early in the annotation retraining process.
The Active Learning methods we employed come from three different families: Score methods, distance based methods and adaptive methods. In addition to methods from literature, we introduced two score methods and utilized known distance measures in a novel way for our purposes. Furthermore, we adapted predictive methods to our use case.
The challenge we faced in the project set up in the K.Rex project was a double black box scenario. While we did not have direct access to data and machine learning models in the K.Rex project, we experimented with methods that have direct access to data and machine learning models in an own classification set up. We found that we do not have a disadvantage when dealing in a double black box scenario. By employing and developing methods that work with predictions we have maximum flexibility when it comes to other classification tasks and models. Our methods are applicable to all classifiers that output a prediction vector. Our experiments were done over pools of unlabeled samples. We note that the score methods and adaptive methods are applicable for stream based scenarios as well.
Under the relative improvement score we saw a very strong performance of distance based methods on the public data and a strong performance of the Oracle Human method and

the Mutual Uniform method on the real data. In general, the usage of Active Learning methods can reduce the amount of needed training samples drastically. For the public data the Vector Norm method led to a 31% higher accuracy than the Entropy method which is known to outperform random sample selection itself [2, 7]. The Vector Norm method is based on the intuition that a diverse training set leads to an increase in performance.

For future work diversity based approaches can be extended to the feature space of the machine learning model. With feature space we mean the model's internal representation of samples. One possibility is to use the flattened vector which is used as an input for the dense layers that do the classification. This vector is for example the flattened output of the last inception layers in the Inception model we employed. Based on the flattened vectors and a vector distance measure we can chose a diverse set of samples just as we did based on the prediction vectors in Section 4.3.

Further future work could include reusing the reinforcement oracle from Section 4.4.2 for different machine learning models on different data. An already trained oracle could make better decisions and lead to a better performance of the different models. Forbidden knowledge transfer from one case data to another case data is prevented since the oracle never sees the data but only the predictions made by the model.

In general, we can conclude that Active Learning methods should always be employed when it is necessary to annotate samples. While the usage of reinforcement learning is not ultimately justified considering the complexity of such an approach, we see great potential for such methods when it comes to reusing oracles for other data sets and models. For diversity and distance based approaches we found that traditional distance measures like Cosine and Euclidean distance work better than novel distance measures like the contextual diversity score. Even computationally inexpensive methods like score calculations on individual prediction vectors save annotation effort and should always be used for training data creation.

Bibliography

[1] Österreichische Forschungsförderungsgesellschaft. Knowledge recognition for evidence extraction, 2020.

[2] Burr Settles. Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison, 2009.

[3] Yazhou Yang and Marco Loog. Active learning using uncertainty information. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 2646–2651, 12 2016.

[4] Ozan Sener and Silvio Savarese. Active learning for convolutional neural networks: A core-set approach. In *International Conference on Learning Representations*, 2018.

[5] Yaqing Wang, Quanming Yao, James T. Kwok, and Lionel M. Ni. Generalizing from a few examples: A survey on few-shot learning. *ACM Comput. Surv.*, 53(3), June 2020.

[6] Mark Woodward and Chelsea Finn. Active one-shot learning. *CoRR*, abs/1702.06559, 2017.

[7] Pengcheng Li, Jinfeng Yi, and Lijun Zhang. Query-efficient black-box attack by active learning. *CoRR*, abs/1809.04913, 2018.

[8] Neil Rubens, Vera Sheinman, Ryota Tomioka, and Masashi Sugiyama. Active learning in black-box settings. *Austrian Journal of Statistics*, 40:125–135, 2 2016.

[9] Donggeun Yoo and In So Kweon. Learning loss for active learning, 2019.

[10] 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.

[11] Sharat Agarwal, Himanshu Arora, Saket Anand, and Chetan Arora. Contextual diversity for active learning, 2020.

[12] Simon Tong and Daphne Koller. Support vector machine active learning with applications to text classification. *J. Mach. Learn. Res.*, 2:45–66, March 2002.

[13] Claude E. Shannon. A mathematical theory of communication. *Bell Syst. Tech. J.*, 27(3):379–423, 1948.

[14] H. Akaike. A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, 19(6):716–723, 1974.

[15] Gideon Schwarz. Estimating the Dimension of a Model. *The Annals of Statistics*, 6(2):461 – 464, 1978.

[16] Leandro Pardo. Statistical inference based on divergence measures. 01 2005.

[17] J. Lin. Divergence measures based on the shannon entropy. *IEEE Transactions on Information Theory*, 37(1):145–151, 1991.

[18] S. Kullback and R. A. Leibler. On Information and Sufficiency. *The Annals of Mathematical Statistics*, 22(1):79 – 86, 1951.

[19] Roxy Peck, Chris Olsen, and Jay L. Devore. *Introduction to Statistics and Data Analysis (with ThomsonNOW Printed Access Card)*. Duxbury Press, USA, 2007.

[20] W. R THOMPSON. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3-4):285–294, 1933.

[21] Disi Ji, IV Logan, Robert L., Padhraic Smyth, and Mark Steyvers. Active Bayesian Assessment for Black-Box Classifiers. *arXiv e-prints*, page arXiv:2002.06532, February 2020.

[22] Yuriy Sverchkov and M. Craven. A review of active learning approaches to experimental design for uncovering biological networks. *PLoS Computational Biology*, 13, 2017.

[23] *An Optimal Screening Experiment*, chapter 2, pages 9–45. John Wiley and Sons, Ltd, 2011.

[24] Tyler Lu, David Pal, and Martin Pal. Contextual multi-armed bandits. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9, pages 485–492, 2010.

[25] Djallel Bouneffouf, Romain Laroche, Tanguy Urvoy, Raphael Feraud, and Robin Allesiardo. Contextual bandit for active learning: Active thompson sampling. In Chu Kiong Loo, Keem Siah Yap, Kok Wai Wong, Andrew Teoh, and Kaizhu Huang, editors, *Neural Information Processing*, pages 405–412, Cham, 2014. Springer International Publishing.

[26] David Cortes. Adapting multi-armed bandits policies to contextual bandits scenarios. *CoRR*, abs/1811.04383, 2018.

[27] Hussein Mozannar and David Sontag. Consistent estimators for learning to defer to an expert. *arXiv e-prints*, page arXiv:2006.01862, 2020.

[28] Chenguang Wang, Laura Chiticariu, and Yunyao Li. Active learning for black-box semantic role labeling with neural factors. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 2908–2914, 2017.

[29] Sanjoy Dasgupta, Daniel Hsu, Stefanos Poulis, and Xiaojin Zhu. Teaching a black-box learner. volume 97 of *Proceedings of Machine Learning Research*, pages 1547–1555, Long Beach, California, USA, 6 2019. PMLR.

[30] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[31] Bhavya Ghai, Q. Vera Liao, Yunfeng Zhang, Rachel Bellamy, and Klaus Mueller. Explainable active learning (xal): An empirical study of how local explanations impact annotator experience. *arXiv e-prints*, page arXiv:2001.09219, 2020.

[32] Pannaga Shivaswamy and Thorsten Joachims. Coactive learning. *Journal of Artificial Intelligence Research*, 53:1–40, 05 2015.

[33] Marco Ribeiro, Sameer Singh, and Carlos Guestrin. "why should I trust you?": Explaining the predictions of any classifier. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*, pages 97–101, San Diego, California, June 2016. Association for Computational Linguistics.

[34] Marc Peter Deisenroth, A. Aldo Faisal, and Cheng Soon Ong. *Mathematics for Machine Learning*. Cambridge University Press, 2020.

[35] Alexander Jung. A gentle introduction to supervised machine learning. *CoRR*, abs/1805.05052, 2018.

[36] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning. *Proceedings of the IEEE*, PP:1–34, 07 2020.

[37] M. Hussain, Jordan J. Bird, and D. R. Faria. A study on cnn transfer learning for image classification. In *UKCI*, 2018.

[38] Zhou Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: From error visibility to structural similarity. *Trans. Img. Proc.*, 13(4):600–612, April 2004.

[39] Amit Singhal and I. Google. Modern information retrieval: A brief overview. *IEEE Data Engineering Bulletin*, 24, 01 2001.

[40] Yaroslav Shitov. Column subset selection is np-complete, 2017.

[41] D. Lewis, G. Agam, S. Argamon, O. Frieder, D. Grossman, and J. Heard. Building a test collection for complex document information processing. *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, 2006.

[42] Jayant Kumar, Peng Ye, and David Doermann. Structural similarity for document image classification and retrieval. *Pattern Recognition Letters*, 43:119–126, 07 2014.

[43] Yann LeCun, Patrick Haffner, Léon Bottou, and Yoshua Bengio. Object recognition with gradient-based learning. In *Shape, Contour and Grouping in Computer Vision*, page 319. Springer-Verlag, 1999.

[44] Christian Szegedy, V. Vanhoucke, S. Ioffe, Jonathon Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826, 2016.

[45] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. pages 1–9, 06 2015.

[46] J. Kiefer and J. Wolfowitz. Stochastic Estimation of the Maximum of a Regression Function. *The Annals of Mathematical Statistics*, 23(3):462 – 466, 1952.

[47] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.

[48] Andrew W. Senior, Georg Heigold, Marc'Aurelio Ranzato, and Ke Yang. An empirical study of learning rates in deep neural networks for speech recognition. In *ICASSP*, pages 6724–6728. IEEE, 2013.

[49] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[50] Lutz Prechelt. *Early Stopping - But When?*, pages 55–69. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998.

[51] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. *Int. J. Comput. Vision*, 115(3):211–252, 2015.

[52] Rich Caruana, Steve Lawrence, and Lee Giles. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. In *Proceedings of the 13th International Conference on Neural Information Processing Systems*, NIPS'00, page 381–387, Cambridge, MA, USA, 2000. MIT Press.