Active Learning in the Medical Domain – an Innovative Cluster Based Uncertainty Sampling Approach

Abstract

Active learning is a machine learning paradigm where instances are selectively/straitlacedly sampled, aiming to maximize the model's performance while minimizing labeling costs. These sampled instances are often selected solely based on some uncertainty measure. However, this approach might limit the labeling process to a group of instances that are not necessarily representative of the entire feature space. This paper suggests a sampling approach which considers the uncertainty of the instances, while aiming to sample from the entirety of the provided data. This will potentially lead to better generalization than traditional greedy uncertainty sampling methods. This approach is implemented by clustering the instances and forcing the sampling procedure to wisely select instances across all clusters. The approach was examined on images data from the medical field, containing chest X-Ray images of Pneumonia-infected and healthy lungs. The results point out that the suggested approach is superior to several baseline methods in detecting infection (i.e. positive instances) during a specific iteration of the AL pipeline. However, the insights do not provide sufficient evidence to support that the approach indeed outperforms said baselines, according to the multiple evaluation metrics tested.

Introduction

As known, training a supervised machine learning model requires labeled data. However, in some cases, obtaining such labels can be both time and resource consuming. This is particularly true in the medical domain, since this process requires the expertise of doctors or specialists. This paper focuses on solutions for a classification problem within this domain. A procedure which aims to handle said issue is Active Learning (AL). This procedure attempts to select instances which would be most beneficial to the model's learning, and thus helps to strategically label the instances.

In recent literature, there are several main approaches for AL (B. Settles, 2009). Let us briefly discuss some of them below:

- **Membership Query Synthesis:** the learner may request labels for any unlabeled instance in the input space, including instances that the learner generates, or "synthesizes", by itself.
- **Stream-Based Selective Sampling:** under the assumption that obtaining unlabeled instances is inexpensive, the learner receives one instance at a time (a "stream"), and may request its label.
- **Pool-Based Sampling:** given a small pool of labeled instances and a large pool of unlabeled ones, the learner samples unlabeled instances whose labels would be most beneficial to its learning, based on some informativeness metrics.

In our project, we focus on the Pool-Based approach. Within this approach, there are multiple sampling methods commonly in use:

- **Uncertainty Sampling:** sampling unlabeled instances which the learner is least confident about. That can be done by calculating some uncertainty measure for all unlabeled instances, and sampling those with the highest measure.
- **Query By Committee:** given a group, or "committee" of ML models, each model predicts the labels of all unlabeled instances. Then, the instances whose labeling was most disagreed on by the committee, are sampled.

In this paper, we choose to focus on the uncertainty sampling method, build upon it and attempt to improve it. When utilizing this sampling method, as the name suggests, it greedily selects instances solely based on the uncertainty measure. However, following this greedy approach might lead to sample only instances with close proximity in the feature space. In other words, this might lead to sampling mostly similar instances. Therefore, the learner will not be exposed to the entire variety of instances during the training process. As a result, it will possess poor generalization capabilities.

In this paper, we suggest a sampling method that aims to select instances with high uncertainty, while forcing the learner to query instances from the entirety of the feature space.

Methodology

As mentioned, this paper suggests a solution for a binary classification problem within the medical domain. Let us introduce below the specific data we work with, as well as the algorithmic approach for the classification task.

Dataset

We work with the "Chest X-Ray Images (Pneumonia)" dataset from Kaggle, after slightly processing it. The processed dataset contains 2,500 grayscale images, equally divided among two classes:

- Normal (chest x-ray contains healthy lungs)
- Pneumonia (chest x-ray contains sick lungs)

For the data processing details, as well as links to data, please refer to the Dataset appendix.

Model

As part of the implementation of the binary image classifier, a Neural Network model was utilized. The architecture of the network was constructed as a concatenation of the two following networks:

1. Convolutional Neural Network (CNN) - containing all convolutional layers of the SOTA model ResNet50[1], along with its pretrained parameters. This section of the model receives 224x224 RGB images as input, and returns a 2048-dimension vector as output (an average pooling vector).
2. Fully Connected Neural Network (FCNN) - containing the following layers:
    - Input layer - the output of the CNN.
    - Embedding layer - vector embedding of size D (hyperparameter), representing the input image.
    - Output layer - the prediction probabilities for each label.

To summarize, our model receives a 224x224 RGB image as input and returns the embedding representation of the image (of size D), as well as its prediction probabilities for each label.

[1] Since image classification is quite a challenging task, we wanted to utilize an existing architecture which proved its capabilities in the past, and adapt it to our problem.

Cluster Uncertainty Sampling

Let us introduce the steps for our sampling method below:

1. Extract embedding vectors and prediction probabilities: For each unlabeled instance, denoted by x_i, use our model, denoted by f, to extract the following:

- The embedding representation of x_i, denoted by $e_i = f_{embed}(x_i)$
- The prediction probabilities vector of x_i, denoted by $p_i = f_{prob}(x_i)$

2. Calculate instance uncertainties: Calculate the uncertainty of instance x_i as the entropy of p_i, denoted by $u_i = -\sum_{j=1}^{n} p_i^{(j)} \log p_i^{(j)}$, where $p_i^{(j)}$ is the prediction probability of instance x_i belonging to class j, and n is the total number of classes.

3. Cluster the instances: Use the K-Means algorithm to partition the embedding vectors $\{e_1, \dots, e_m\}$ into k clusters $\{C_1, \dots, C_k\}$, where k is a hyperparameter.

4. Calculate cluster-level metrics: For each cluster C_j, calculate the following metrics:

- Uncertainty Score – The average uncertainty of all the cluster's members, denoted by $U(C_j) = \frac{1}{|C_j|} \sum_{x_i \in C_j} u_i$
- Size – The total number of members in the cluster, denoted by $S(C_j) = |C_j|$

5. Calculate cluster overall score: For each cluster, calculate its overall score as a weighted combination of its uncertainty score and its size, denoted by $O(C_j) = \alpha * U(C_j) + (1 - \alpha) * S(C_j)$, where α is a weighting hyperparameter.

6. Normalize cluster overall scores: Apply the $SoftMax$ function to the cluster overall scores vector in order to obtain normalized scores, denoted by $\hat{O}(C_j) = \frac{\exp(O(C_j))}{\sum_{l=1}^{k} \exp(O(C_l))}$

7. Determine the sampling budget of each cluster: Let B represent the total sampling budget for the current iteration. For each cluster C_j, calculate its budget, denoted by $B(C_j)$, as follows: $B(C_j) = \min\left(\hat{O}(C_j) * B,\ S(C_j)\right)$.

8. Select instances to label: For each cluster C_j, sample the $B(C_j)$ members with the highest uncertainties.

Remarks

1. While we choose to measure the instance uncertainties using the Entropy function, any uncertainty measure is applicable.

2. When calculating the overall score, both the cluster's uncertainty score and size metrics are scaled to the range [0, 1], using MinMax Scaling. This process eliminates the differences in the metrics' original scales, since those differences might result in a bias towards the larger scaled metric.

3. When calculating each cluster's sampling budget, the product (first element of the minimum) might not be an integer. However, since we sample instances, the budget must be one. As a result, we round down the product to the nearest one. This solves said issue while also avoid exceeding the current iteration's budget.

4. It is possible that the total sum of the clusters' budgets will be strictly less than the current iteration's budget (due to the round down or the minimum statement). In such case, the unallocated budget is distributed between clusters with available budget (i.e. cluster's product is smaller than its size), in decreasing order of their overall scores. Specifically, each ordered cluster is assigned the maximum possible amount from the unallocated budget before proceeding to the next one. This ensures that in each iteration, the total budget is indeed sampled.

Experiments

In this section, we first present the hyperparameter tuning process for our approach, along with its results. Then, with the obtained hyperparameters, we showcase and analyze the method's performance while comparing it to several baseline sampling methods.

In all experiments below, the AL pipeline uses all instances within the dataset (2,500), in the following manner: First, 20% of the instances are randomly assigned to the test set (500), 20% are randomly assigned to the initial training set (500), while the remaining 60% are assigned to the available pool of unlabeled instances (1,500). Then, the AL pipeline is run for exactly 10 iterations, with a budget of 150 instances (10% of the initial available pool) in every iteration, meaning all available pool instances are labeled by the end of the last iteration. In each iteration, the model is trained from scratch on the current training set, with 3 training epochs. Note that in order to fine tune the model to the dataset, the pretrained CNN's parameters are frozen during the training process, and only the FCNN parameters are updated.

Hyperparameter Tuning

Let us conduct several experiments in order to find the best set of hyperparameters, as mentioned in the Methodology section. Recall that our approach contains three hyperparameters:

- k - the number of clusters
- α - the weighting parameter for the overall score calculation
- D - the embedding dimension that the input images are embedded to

First, we set D = 1024, as a reasonable embedding size. Then, while D remains constant, we perform grid search in order to optimize for the best performing (k, α) pair.

- For the k parameter, we examine the following values: 2, 4, 6, 8, 10
- For the α parameter, we examine the following values: 0.0, 0.2, 0.4, 0.6, 0.8, 1.0

Moreover, the goal of this procedure is finding the parameters that lead to the sampling of the most informative unlabeled instances. For that reason, we compare the accuracies of the model on the test set after iteration 5 of the AL pipeline (training on 62.5% of the total train set), rather than after the last iteration (where the model trains on the entire train set, regardless of the hyperparameter values).

As can be seen in Fig 1 under the Plots appendix, the highest accuracy (0.95) is received for $k = 6$ and $\alpha = 0.4$. Thus, we set those hyperparameters with said values.

Now, we search for the best performing embedding dimension D. Since the initial value (1024) is a relatively high embedding size, let us explore dimensions lower than that. In particular, we examine the following values: 256, 512, 768, 1024.

As can be seen in Fig 2 under the Plots appendix, the highest accuracy is received for $D = 1024$. That is, we decide to stick with the initial embedding dimension, which yields an accuracy of 0.95 in iteration 5 of the AL pipeline, along with the tuned (k, α) pair.

Therefore, to summarize, the tuned hyperparameter values are $\boldsymbol{k = 6}$, $\boldsymbol{\alpha = 0.4}$, $\boldsymbol{D = 1024}$.

Performance Analysis

After obtaining the tuned hyperparameters, let us analyze the performance of our approach. We use several metrics to evaluate the suggested sampling method, and compare them with the following sampling methods, as baselines:

- Random Sampling – selecting instances to label at random.
- Entropy Uncertainty Sampling – calculating the entropy of all unlabeled instances, and selecting the ones with the highest entropy.
- "MinMax" Uncertainty Sampling – finding the prediction probability of the label that the model is most certain about, and selecting the instances with lowest said probability.

First, we find the accuracies of all sampling methods across all iterations of the AL pipeline. The insights from Fig 3 do not appear to be very indicative of any relations between the performance of the sampling methods. That is, the accuracy seems to be noisy and not form any apparent trend across the different iterations. Thus, the insights do not sufficiently allow us to draw any conclusions regarding the suggested method's performance compared to the baselines.

Secondly, in order to further investigate the model's performance across the iterations, let us discuss the TPR and TNR evaluation metrics. From Fig 4 and Fig 5, again, the results do not seem to be coherent, and do not provide insights regarding the wellness of the suggested method.

Since the evaluation metric values vastly change between the pipeline's iterations, we focus on a specific iteration. We decided to focus on iteration number five again, in which the model is trained on 62.5% of the training set, due to it being roughly the middle-way iteration in the procedure. This demonstrates a real-world active learning use case, where the model is trained only on a subset of the available training set due to high labeling costs.

Let us analyze the confusion matrices received in iteration 5 of the AL pipeline[2]. From Fig 6 it appears that the Cluster Uncertainty sampling method received a decent confusion matrix; it received the lowest number of False Negatives among all baselines, yet the highest number of False Positives.

[2] We traditionally refer to the "Pneumonia infected" label as Positive, and refer to the Normal, "healthy" label, as Negative.

From Fig 7, it appears that the Cluster Uncertainty method yielded the highest TPR, meaning that it correctly identified the most positives across all sampling methods. However, it also yielded the lowest TNR, meaning that it correctly identified the least Negatives across all methods. Lastly, from Fig 8, it seems that the suggested approach is outperformed by every baseline method, in terms of F1 score in iteration 5 of the AL pipeline.

Discussion

Limitations

As mentioned in the Experiments section, when evaluating the suggested sampling method and comparing it to the discussed baselines, the evaluation metrics fluctuate quite heavily between iterations. This does not allow us to draw any strong conclusions regarding the comparative wellness of our method across all iterations. In future work, given more time and computation resources, we will consider using a more extensive training process of the model (e.g. more training data and epochs). In addition, we would run the experiments with multiple randomness seeds to minimize the effect of randomness on the results. These measures can assist in improving the results' stability, and hopefully provide better insights to the method's performance. Moreover, we will consider other architectures for the model, and attempt to generalize our approach to non-image data as well.

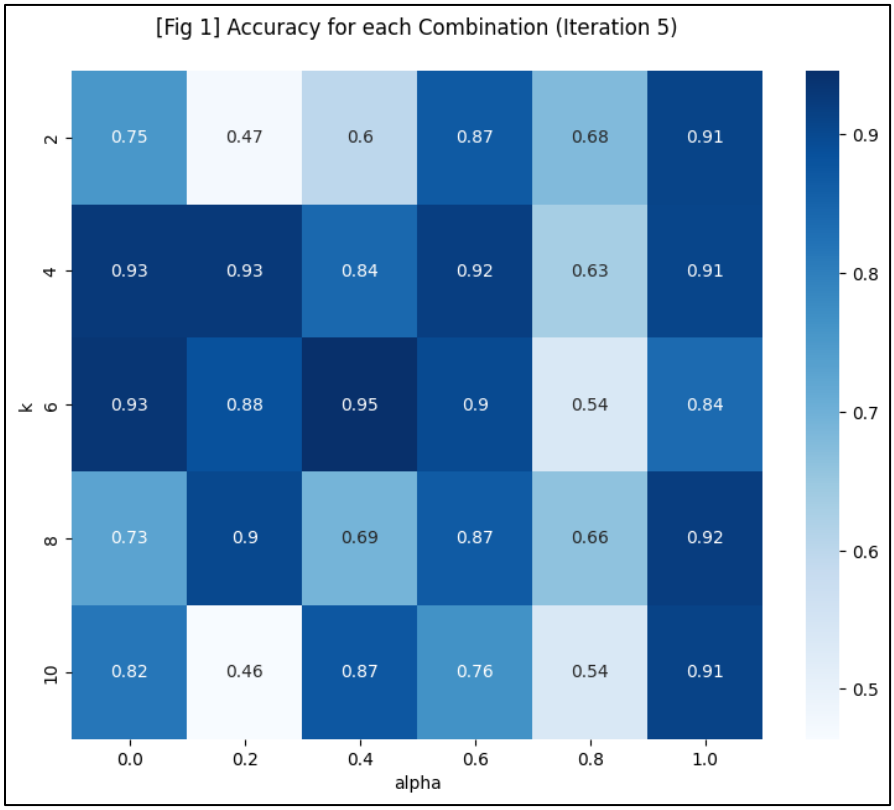
Key Findings

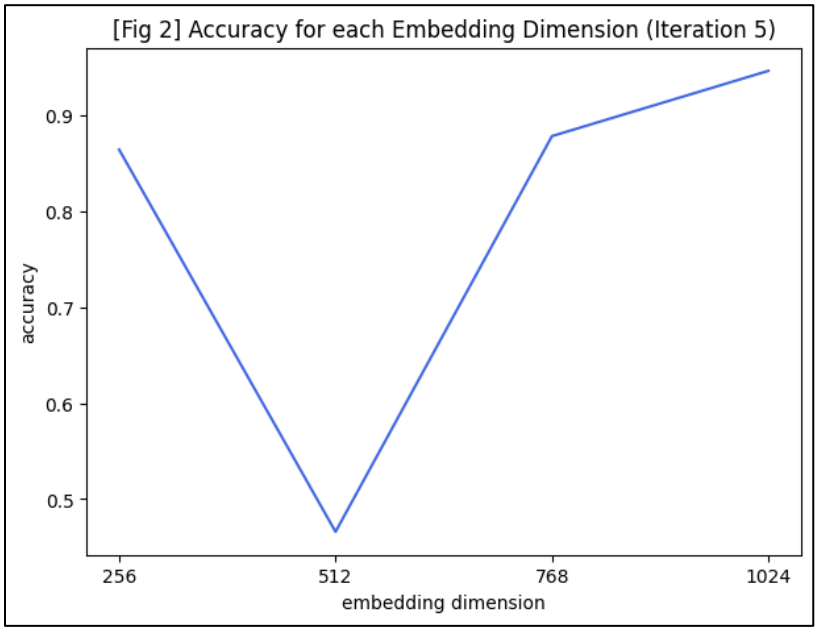
Although no significant conclusions were provided by the results across all iterations, some interesting insights were found when focusing on a single iteration. In the iteration that was examined (5), when the model was trained on 62.5% of the training set, the suggested method outperformed all baselines in terms of TPR. This highlights that when using our method, the model's ability to detect positive instances (Pneumonia-infected lungs) is superior in that iteration. Note that for the medical domain in general, the ability to detect positive instances is important, since it is often safer to wrongly diagnose a healthy person for a disease, rather mistakenly diagnose a sick person as healthy. However, on the other hand, the suggested method was outperformed by all baselines in terms of the TNR and F1 score evaluation metrics in that iteration.

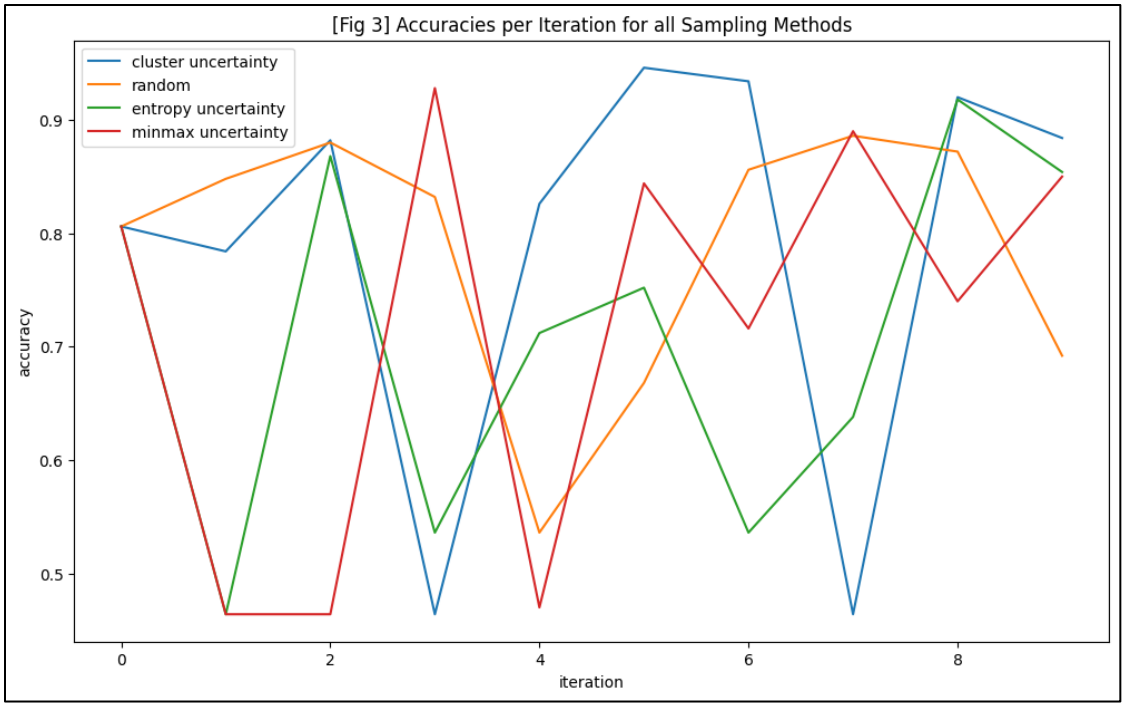
Appendix

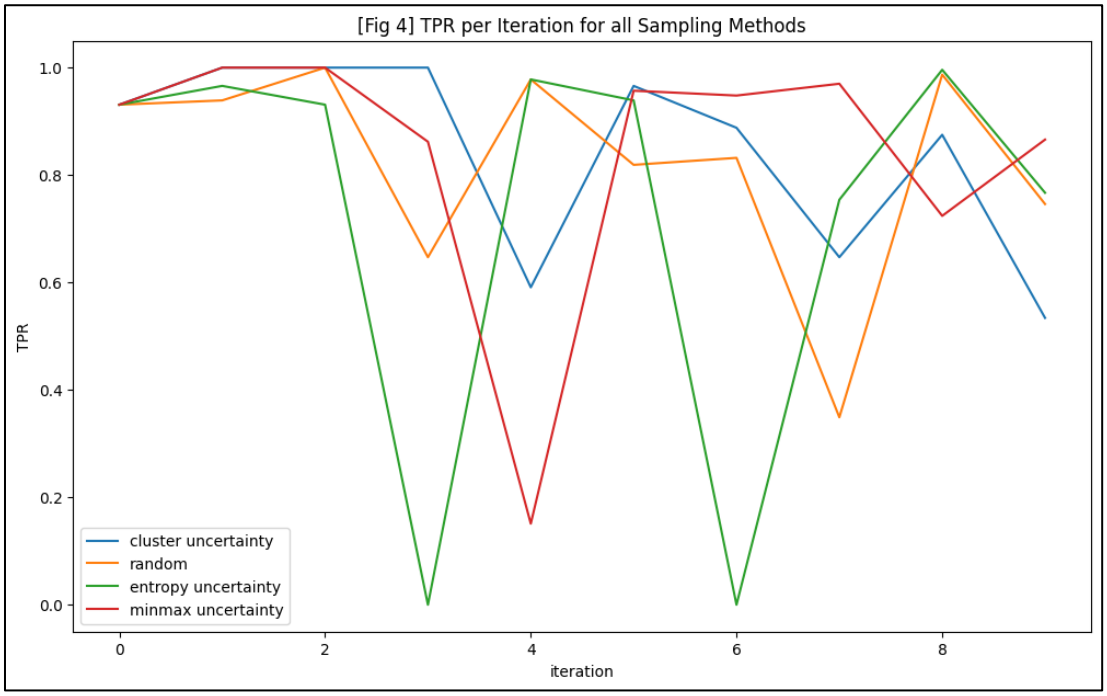
Plots

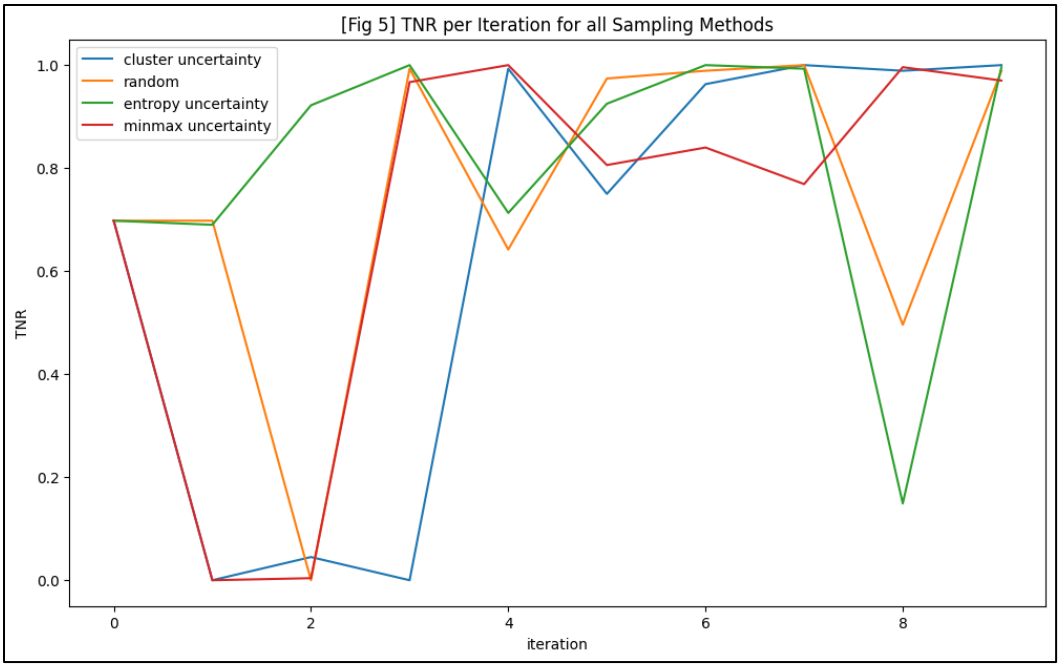
Below are all figures discussed in the main paper, with the figure number displayed in the title:

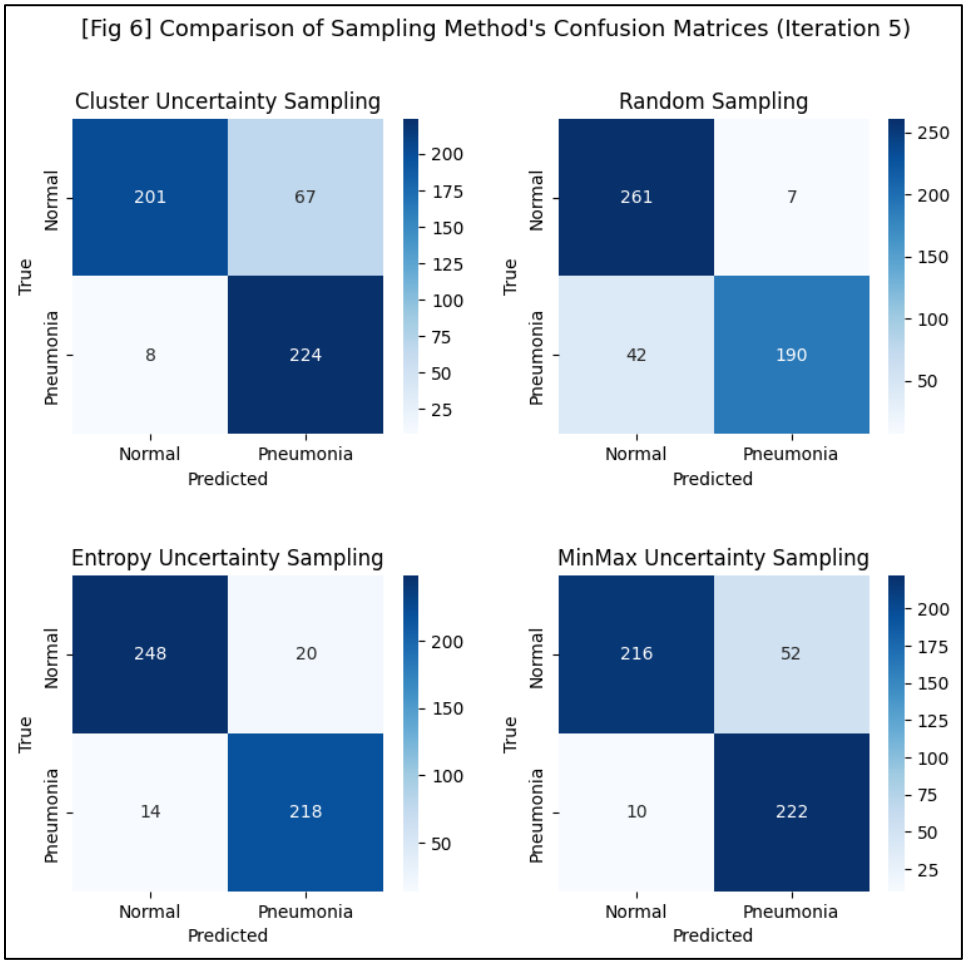


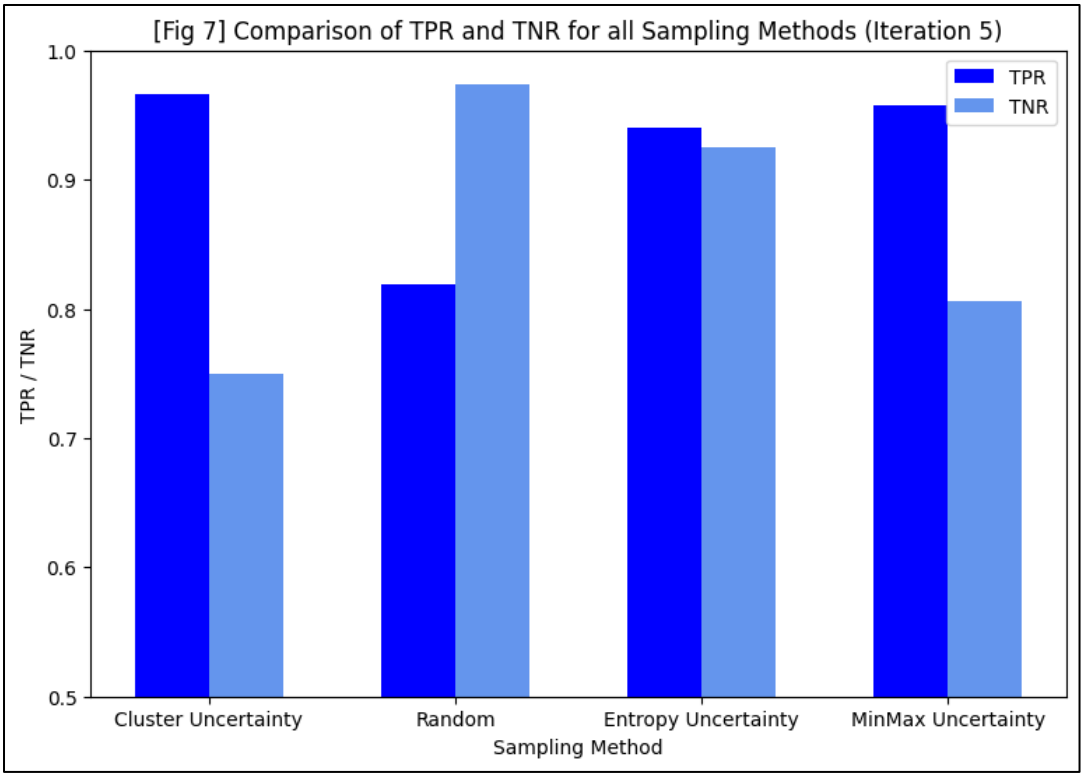


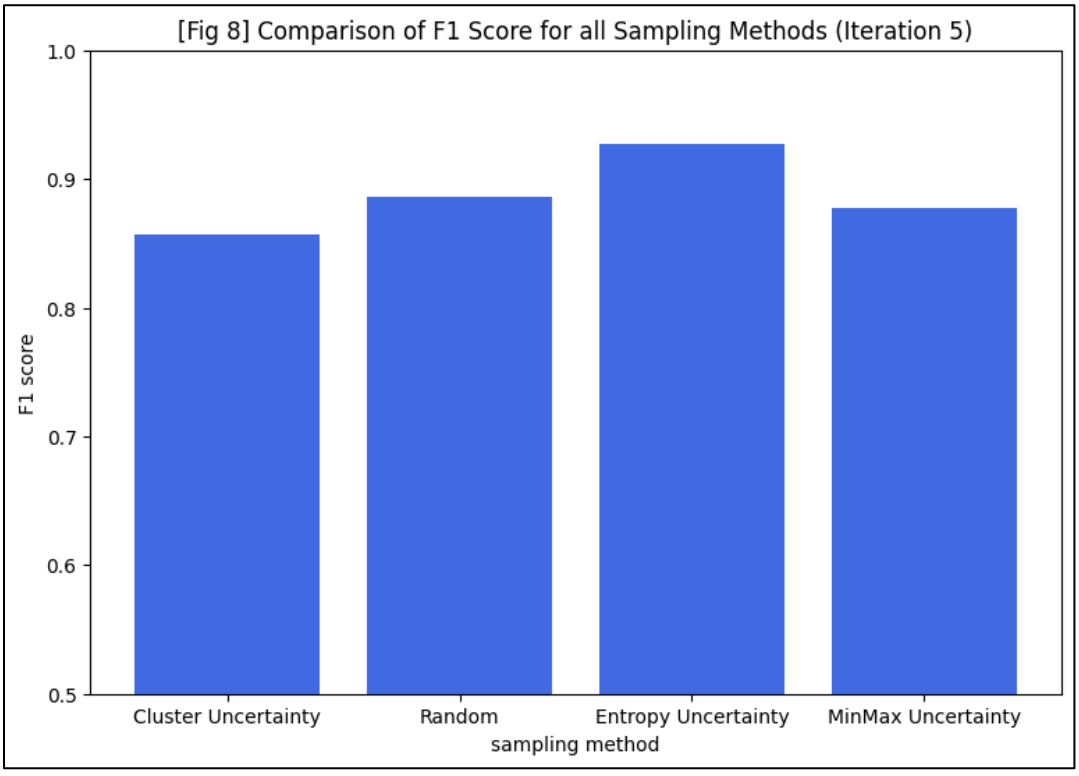

[Fig 3] Accuracies per Iteration for all Sampling Methods
cluster uncertainty
random
entropy uncertainty
minmax uncertainty
accuracy
iteration
0.5
0.6
0.7
0.8
0.9
0
2
4
6
8


[Fig 4] TPR per Iteration for all Sampling Methods
cluster uncertainty
random
entropy uncertainty
minmax uncertainty
TPR
iteration
0.0
0.2
0.4
0.6
0.8
1.0
0
2
4
6
8


[Fig 5] TNR per Iteration for all Sampling Methods
cluster uncertainty
random
entropy uncertainty
minmax uncertainty
TNR
iteration
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8


[Fig 6] Comparison of Sampling Method's Confusion Matrices (Iteration 5)
Cluster Uncertainty Sampling
201
67
8
224
Normal
Pneumonia
True
Predicted
Random Sampling
261
7
42
190
Entropy Uncertainty Sampling
248
20
14
218
MinMax Uncertainty Sampling
216
52
10
222


[Fig 7] Comparison of TPR and TNR for all Sampling Methods (Iteration 5)
TPR
TNR
TPR / TNR
1.0
0.9
0.8
0.7
0.6
0.5
Cluster Uncertainty
Random
Entropy Uncertainty
MinMax Uncertainty
Sampling Method


[Fig 8] Comparison of F1 Score for all Sampling Methods (Iteration 5)
F1 score
1.0
0.9
0.8
0.7
0.6
0.5
Cluster Uncertainty
Random
Entropy Uncertainty
MinMax Uncertainty
sampling method

Dataset

As mentioned in the main paper, the full dataset used is “Chest X-Ray Images (Pneumonia)” from Kaggle, which can be found in the following link. This dataset contains 5,863 images, divided between two classes. However, due to limited time and limited computational resources, only a subset of the full dataset was used during the experiments in the paper.

When extracting the processed dataset for the project, we focused only on the train set provided with the data, and from it we selected 1,250 images from each class. This resulted in a balanced dataset with a total of 2,500 images. The processed dataset can be found in the following Google Drive link. Note that this dataset is not split into train, test and available pool sets in advance. Rather, the splitting is done online (within the code).

Sources

First, when conducting the literature review in the Introduction section, the main article that was used as an information source is “Active Learning Literature Survey”, by Burr Settles, 2009.

Additionally, the approach discussed in the paper was inspired by the approach presented in the article “The Battleship Approach to the Low Resource Entity Matching Problem”, by Bar Genossar et. al, 2023.

GitHub Repository

The full code implemented for this paper, along with the necessary running instructions, can be found in the project’s GitHub repository, in the following link.

Note that the entire team worked on the code together, and quite continuously. Then, when finished, the final code was uploaded to the GitHub repository at once. Therefore, no significant commit history can be seen in the repository.