Probabilistic Active Learning for Active Class Selection

Daniel Kottke
Intelligent Embedded Systems
University Kassel, Germany
daniel.kottke@uni-kassel.de

Georg Krempl
Knowledge Management & Discovery
University Magdeburg, Germany
georg.krempl@ovgu.de

Marianne Stecklina **Cornelius Styp v. Rekowski** **Tim Sabsch** **Tuan Pham Minh**
University Magdeburg, Germany
marianne.stecklina, cornelius.styp, tim.sabsch, tuan.pham@ovgu.de

Matthias Deliano
Leibniz Institute for
Neurobiology, Magdeburg
deliano@lin-magdeburg.de

Myra Spiliopoulou
KMD Group
University Magdeburg
myra@cs.uni-magdeburg.de

Bernhard Sick
IES Group
University Kassel
bsick@uni-kassel.de

Abstract

In machine learning, active class selection (ACS) algorithms aim to actively select a class and ask the oracle to provide an instance for that class to optimize a classifier's performance while minimizing the number of requests. In this paper, we propose a new algorithm (PAL-ACS) that transforms the ACS problem into an active learning task by introducing pseudo instances. These are used to estimate the usefulness of an upcoming instance for each class using the performance gain model from probabilistic active learning. Our experimental evaluation (on synthetic and real data) shows the advantages of our algorithm compared to state-of-the-art algorithms. It effectively prefers the sampling of difficult classes and thereby improves the classification performance.

1 Introduction

Many classification systems found in application areas such as economy, medical research or neurobiology require human labeling effort during training. As this is time-consuming and expensive, the field of active learning (AL) emerged [15]. Here, the aim is to actively choose only the most informative instances from a large pool of unlabeled data and successively request their label. As a result, good classification performance is reached with less training instances compared to passively feeding arbitrary instances to the classifier.

In this article, we address a related field called active class selection (ACS) [8]. Instead of selecting an unlabeled instance and acquiring its label, ACS methods request a yet unseen instance by selecting its class. On the one hand, the degree of freedom is much smaller in ACS compared to AL as there are normally less classes than instances. On the other hand, less information is available to decide what is beneficial for training, e.g. there are no unlabeled instances to approximate the data distribution.

Why is ACS a topic worth researching? A vivid example for the application of ACS is the training of brain computer interfaces for motoric prostheses. To train such a prosthesis, an impaired patient has to imagine motoric movements [4], for example finger movements, while the brain activity

30th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.

is recorded. Fig. 1 shows different learning stages of such an exemplary ACS process. In the beginning, the algorithm only knows the number of classes (fingers). If certain classes (fingers) are hard to distinguish in the data (here class 1), learning should focus on these classes. By requesting the patient to generate more training instances of these classes instead of spending time on already learned classes, a good classification performance is achieved earlier – an achievement that enables the patient to perform otherwise impossible tasks [4].

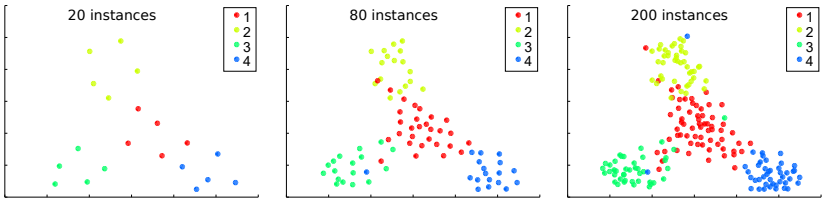


Figure 1: t-SNE plot for different learning stages in an ACS sampling process

As visualized by this example, ACS is useful whenever classes vary in difficulty. We contribute a new method to the field of ACS that is able to identify the difficulty of classes. The core idea is to transform the ACS task into an active learning problem which enables the applicability of well-understood active learning paradigms: In each step, we simulate the generation of instances (called pseudo instances). Next, we take the performance gain function used in probabilistic active learning [5] to determine the expected usefulness of a new instance request for every class. Finally, we request an instance from the class with highest usefulness. This is repeated until a maximal number of instances (budget) is reached.

The rest of the article is structured as follows. In Sec. 2, we discuss the literature on active class selection, followed by our new method PAL-ACS. We provide a pseudo code and discuss properties of the approach by using an example. After an evaluation on multiple datasets in Sec. 4, we finally conclude our work.

2 Related Work

Active classification systems have the ability to request relevant information from external sources. With respect to the type of requested information, different approaches are distinguished [2]. The most intensively researched ones actively select instances for labeling from an oracle. The aim of these so-called active learning methods is to select those instances whose labels will improve the classification performance the most [15]. Scope of this paper is the inverse setting of active learning, which is called active class selection (ACS) [14]: Here, the active component is able to select a class from which subsequently an unknown instance (feature vector) is generated.

The idea of ACS is to distribute the number of instances per class such that a certain level of classification performance is reached with the lowest number of requested instances [2, p. 29]. The work presented in [10] (see also [9]) mentions different techniques to determine this class distribution for acquisition chunks. First, Lomasky et al. [10] propose to use a *uniform* distribution and the *Original Proportion* (that usually is not known) as baselines. Furthermore, they perform what they call f-fold cross validation on the already seen chunks to use the results for the next chunk: The approach *Inverse* distributes the information according to the inverse of the class accuracy. An extension of this is called *Accuracy Improvement*. It distributes the values according to the accuracy difference between the two most current chunks. The *Redistricting* method counts the number of labels that have been flipped (these instances are marked as redistricted) by adding the most recent chunk to the training set. Here, the upcoming instances are distributed with respect to the number of redistricted instances of the true classes.

Wu and Parsons [17] applied the previous algorithms *Inverse* and *Accuracy Improvement* to arousal classification. Later, they extended this article in [16] and improved the approach *Inverse* to be applicable for incremental stream acquisitions along with the addition of a constraint that two consecutive new training examples are from different classes. In her PhD thesis [8], Lomasky extended her work by two more methods: *Risk* estimates the sensitivity of error that is induced by adding

new instances of a certain class, and *Sensitivity* measures the stability of class decisions. As these methods are only mentioned in the PhD thesis, yielding mediocre results therein, we solely consider the former ones in our evaluation.

As mentioned in the introduction, our new method transforms the ACS task into an active learning problem [15] and uses probabilistic active learning [6], which is assigned to the group of decision theoretic methods. Classical decision theoretic approaches simulate the acquisition of every possible label and evaluate their effect on the classification error using an evaluation set [13]. In [3], Chapelle observed that these error reduction estimates have issues with unreliable posterior estimates at the beginning. Thus, he suggests using a beta-prior to shift posterior values with less labeled information towards equal posterior probabilities. Probabilistic active learning [6] reduces the computational complexity of the previous methods by using local statistics (number of nearby labels) to estimate the usefulness of a labeling candidate. This approach models the true posterior probability with a Beta distribution in order to include the reliability of the posterior. Probabilistic active learning has been extended for multi-class problems in [5] and is discussed in Sec. 3.1 in more detail.

3 Our Method

In this section, we propose our new method called *Probabilistic Active Learning for Active Class Selection (PAL-ACS)*. The first subsection gives a detailed description of our algorithm including the necessary background on probabilistic active learning. To support the understanding of the algorithm, we show a visualization of its behavior and provide a pseudo code which can be used for implementing the approach in the second and third subsection.

3.1 Probabilistic Active Learning for Active Class Selection (PAL-ACS)

The main idea of our algorithm is to estimate the gain in classification performance for each class $y \in Y = \{1, \ldots, C\}$ when requesting one additional instance of that class y. Then, we request an instance x^* of the best class y^* and add this new training sample to the training set $\mathcal{L} \leftarrow \mathcal{L} \cup (x^*, y^*)$.

To estimate the expected gain in performance that a label request would probably induce, probabilistic active learning [5] provides an effective tool. Its performance gain function can be calculated at any location in the feature space, without requiring a real unlabeled instance at this location. It only requires local statistics, which typically are labeling counts $\vec{k} = (k_1, \ldots, k_C)$. A common strategy to determine this vector are kernel frequencies [5]. These sum up the similarities from the requested location (x) to every instance from class y.

$$k_i = \mathrm{KFE}(x, \mathcal{L}_i) = \sum_{\{(x',y') \in \mathcal{L}_i\}} \mathrm{sim}(x, x') = \sum_{(x',y') \in \mathcal{L}_i} \exp\left(-\frac{||x - x'||^2}{2\sigma^2}\right) \tag{1}$$

$$\mathcal{L}_i = \{(x, y) \in \mathcal{L} \colon y = i\} \tag{2}$$

In probabilistic active learning [6], it is assumed that the similarity function defines a neighborhood around the requested location x which separates the data into being inside resp. outside the neighborhood. Hence, a newly added label of class y would increase k_y by 1, resp. n new labels increase the corresponding elements in the frequency vector $\vec{k}$ by n. Accordingly, the labeling vector $\vec{l} = (l_1, \ldots, l_C)$, $l_i \in \mathbb{N}$, $\sum l_i = m$ is defined such that the cells contain the number of labels that might be added to that neighborhood [5]. Considering to add $m = 2$ labels to a neighborhood for example (let $C = 3$), the labeling vectors could be $\vec{l} \in \{(2,0,0), (0,2,0), (0,0,2), (1,1,0), (1,0,1), (0,1,1)\}$.

The performance gain function (Eq. 3) [5] for label statistics $\vec{k}$ is defined by subtracting the current expected performance (0 labels added) from the future expected performance (m labels added). This function includes the addition of *multiple* hypothetical labels ($m > 1$). As we only acquire labels successively (one-by-one), we divide the gain by the number of labels m which could be interpreted as the average gain in performance. In our application, the parameter M which sets the upper bound for the so-called local budget has been set to 3 to avoid high computational time.

$$\mathrm{perfGain}\left(\vec{k}, M\right) = \max_{m \leq M} \left(\frac{1}{m}\left(\mathrm{expPerf}\left(\vec{k}, m\right) - \mathrm{expPerf}\left(\vec{k}, 0\right)\right)\right) \tag{3}$$

The expected performance in Eq. 4 is a decision theoretic formulation calculating the expectation values over all possible posterior probabilities $\vec{p}$ and over all possible labeling vectors $\vec{l}$ [5]. The probability of a posterior probability $P(\vec{p} \mid \vec{k})$ to be true given the current label statistics is derived using the likelihood of the multinomial distribution. The probability of a labeling vector $P(\vec{l} \mid \vec{p})$ to be true is directly given by the multinomial distribution. We optimize the performance in terms of accuracy as given in Eq. 5. More details can be found in [5].

$$\text{expPerf}\left(\vec{k}, m\right) = \mathop{\mathbb{E}}_{\vec{p}}\left[\mathop{\mathbb{E}}_{\vec{l}}\left[\text{perf}\left(\vec{k}+\vec{l} \mid \vec{p}\right)\right]\right] \tag{4}$$

$$\text{perf}\left(\vec{k}+\vec{l} \mid \vec{p}\right) = \vec{p}_{\hat{y}} \qquad \hat{y} = \arg\max(\vec{k}+\vec{l}) \tag{5}$$

In contrast to active learning, we do not have access to a pool of unlabeled instances in ACS. Thus, we propose to generate pseudo instances x_p to transform the active class selection problem into an active learning task. We then use the pseudo instances to determine the most beneficial class y^* which is selected according to Eq. 6. Distributing the pseudo instance randomly or equidistant over the whole feature space, we have to add two weights: (1) Similarly to probabilistic active learning, we incorporate an instance's impact on the overall classification performance, i.e. a density weight [7] ($P(x_p \mid \mathcal{L})$). (2) We weight the instance by the probability given that it is assigned to the requested class y to distinguish the classes ($P(x_p \mid \mathcal{L}, y)$). In practice, we use a Monte Carlo approach instead of equidistant sampling as described in Sec. 3.3.

$$y^* = \arg\max_y \left(\sum_{x_p} P(x_p \mid \mathcal{L}) \cdot P(x_p \mid \mathcal{L}_y) \cdot \text{perfGain}(KFE(x_p, (\mathcal{L})_i)) \right) \tag{6}$$

3.2 Characteristics of PAL-ACS and Example

We now discuss PAL-ACS's approach in two exemplary active class selection situations shown in Fig. 2. Both situations are based on a three-class-classification task with a one-dimensional feature space. One class (blue) is well separated from the other two classes (red and green) and can therefore be considered to be easy. Due to an overlap of the other two classes, finding the best decision boundary between them is more difficult.

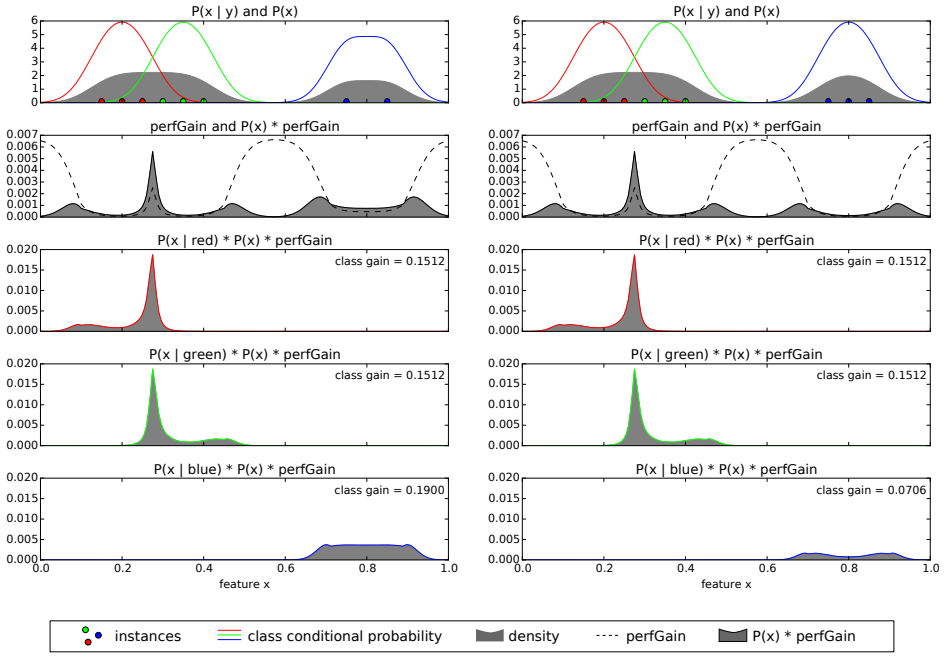


Figure 2: Visualization of *PAL-ACS* for different situations.

The situations shown in the left and right columns are from consecutive selection steps. On the left, 8 instances (3 red, 3 green, 2 blue) have already been acquired, and on the right, there is one additional blue instance. The upper plots show the location of the instances (colored dots on the x-axis) with their corresponding class (red, green, and blue). Furthermore, they show the class conditional distributions in the corresponding color and the density as a gray area. The plots in the second horizontal row show the $\mathrm{perfGain}$ function over the whole feature space as a black dashed line, and the density weighted $\mathrm{perfGain}$ as a solid line with gray area. The lower three plots show the density weighted $\mathrm{perfGain}$ (solid black curve from above) additionally weighted with the corresponding class conditional probabilities which build the final score from Eq. 6. The numbers in the upper right corners represent the sum of the corresponding values. The class with the maximal value is chosen for the next instance generation.

The difference between both snapshots is the smaller number of blue instances on the left. Thus the uncertainty is higher which is underlined by a higher performance gain value. This lack of information is responsible for having an instance of the blue class requested. On the right, all classes are equally well represented (by 3 instances each). Here, the complexity of the decision boundary and the uncertainty in that specific region is responsible for preferring the red, resp. green, class. As discussed in [5], the $\mathrm{perfGain}$ function balances exploration and exploitation by using the number of nearby labels. This also works when using the model for ACS tasks.

3.3 Implementation and Pseudo Code

In Fig. 3, we provide the pseudo code of our approach, starting with the sampling of pseudo instances $\mathcal{X}^p$ in line 3. Especially for high-dimensional data, an equidistant sampling of the whole feature space exceeds computational capabilities. Hence, we use a Monte-Carlo approach in our implementation. From each class, we sample $n_p = 25$ pseudo instances from the corresponding density distribution (line 3). The distribution to sample from is determined by a kernel density estimation similar to the frequency estimate's kernel. In ACS, it is generally assumed that each class is similarly important (albeit not all are necessarily equally difficult). Therefore, we sample the same number of pseudo instances from each class.

1: $n_p \leftarrow 25, \quad M \leftarrow 3, \quad \mathcal{L} \leftarrow \{\}$ ▷ Set initial standard values
2: **while** instance acquisitions left **do**
3: $\quad \mathcal{X}^p \leftarrow \text{SampleFromDensity}(\mathcal{L}, n_p)$ ▷ Sample pseudo instances
4: $\quad$ **for** $i \in \{1, \ldots, |\mathcal{X}^p|\}$ **do** ▷ Calculate pseudo instance's perfGains
5: $\quad\quad k_{i,\cdot} \leftarrow \text{getKVector}(x_i^p, \mathcal{L})$
6: $\quad\quad pg_i \leftarrow \text{perfGain}(\vec{k}_i, M)/|\mathcal{X}^p|$ ▷ Multiply density weight
7: $\quad$ **end for**
8: $\quad$ **for** $y \in \{1, \ldots, C\}$ **do** ▷ Summarize weighted perfGains
9: $\quad\quad g_y \leftarrow 0$
10: $\quad\quad$ **for** $i \in \{1, \ldots, |\mathcal{X}^p|\}$ **do**
11: $\quad\quad\quad g_y \leftarrow g_y + pg_i \cdot k_{i,y}/(\sum_{j=1}^{|\mathcal{X}^p|} k_{j,y})$
12: $\quad\quad$ **end for**
13: $\quad$ **end for**
14: $\quad y^* \leftarrow \arg\max_y(g_y)$ ▷ Select optimal class
15: $\quad x^* \leftarrow \text{requestInstance}(y^*)$
16: $\quad \mathcal{L} \leftarrow \mathcal{L} \cup (x^*, y^*)$
17: **end while**

Figure 3: Pseudo code of the PAL-ACS algorithm.

In the for-loop (lines 4-7), we estimate the kernel frequency vector as defined in Eq. 1 and calculate the corresponding performance gain (see Eq. 3) for each pseudo instance. As all values are generated from the data, each pseudo instance is now equally probable. As we sampled the instance according to the density, the density weight is a simple division by the number of pseudo points. In lines 8-13, we weight this density-weighted performance gain with the class conditional probability and sum all values for each class separately. Finally, we select the best class gain g_y and request a corresponding instance (lines 14-16).

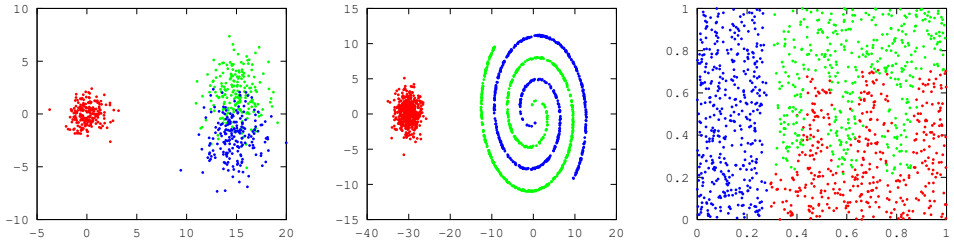
The parameter M of the perfGain function is a parameter of the probabilistic approach that defines the so-called local budget. In case M additional labels are not able to change the classification decision in a neighborhood, the perfGain is zero. This might lead to inconsistencies in the learning process. In our experiments, a value of $M = 3$ was sufficient (higher M means more computation time) as the results with higher M were completely equal. It is also possible to set M to a smaller value but this adds some noise leading to slightly poorer results.

4 Evaluation

In this section, we evaluate the probabilistic active learning for active class selection (PAL-ACS)-approach against other methods on multiple datasets in experimental comparisons. After describing our evaluation setup, we provide learning curves as well as error and sampling proportion tables and discuss the results.

4.1 Evaluation Setup

The methods are evaluated on six different datasets. Thereof, three datasets are synthetic, having one class that is easily distinguishable from the others and two classes with a more complex decision boundary. A visualization of these two-dimensional datasets, called 3Clusters, Spirals and Bars, is given in Fig. 4a–4c. Additionally, we used three real-world datasets from the UCI machine learning repository [1], namely Vehicle, Vertebral Column, and Yeast. For Yeast, we selected five classes for our application: CYT, NUC, ME1, ME2, and ME3. We set the maximum number of learning steps, i.e. the budget depending on the complexity of the datasets to: 60 for 3Clusters, Vertebral, and Yeast, 80 for Vehicle, and 120 for Bars and Spirals.



(a) 3Clusters (b) Spirals (c) Bars

Figure 4: Scatterplots of the synthetic datasets.

As a baseline approach, we implemented the selection strategy *Random* that requests each class with equal probability. Furthermore, we compare against the state-or-the-art approaches *Inverse* and *Redistricting* published in [10].

In a pre-processing step, all features are normalized to a $[0, 1]$ range. For each dataset, we generated 500 random test-training-set combinations (trials). A test set consisting of 50 instances per class is extracted from the data, all remaining instances are used for training. To classify unseen data from the test set, we use a Parzen window classifier [3, 12] with the same kernel used in the kernel frequency estimation. Due to feature normalization, the use of a constant kernel width for all datasets is reasonable, which we set to $\sigma = 0.05$. Error rate is used as performance measure and averaged over the 500 trials.

To ensure that only an algorithm's sampled class distribution influences its classification performance, we decided to use a fixed order of training instances per trial. When an algorithm requests an instance, the first instance of this class is returned. As a consequence, the training data obtained by different algorithms might overlap largely. Consider an example, where one ACS algorithm samples equally while the other samples 40% from class 1 and 2 and 20% from class 3. Although their sampled class distributions differ considerably, 26 of their first 30 acquired instances are completely equal. The fact that the resulting classifiers are therefore similar should be considered when reading the evaluation in the next chapter.

Dataset	Method	phase 1		phase 2		phase 3		phase 4	
		error	win ratio	error	win ratio	error	win ratio	error	win ratio
3Clusters	PAL-ACS	**0.1498**	**40.97%**	**0.1316**	**42.69%**	**0.1215**	**45.85%**	**0.1161**	**47.93%**
	Inverse	0.1543	38.97%	0.1382	33.67%	0.1271	34.28%	0.1206	34.17%
	Redistricting	0.1557	36.10%	0.1418	30.25%	0.1339	31.28%	0.1281	31.69%
	Random	0.1575	35.18%	0.1390	31.13%	0.1294	30.93%	0.1217	32.68%
Bars	PAL-ACS	0.2705	25.97%	0.1773	28.82%	0.1378	32.47%	**0.1177**	**38.22%**
	Inverse	0.2636	31.12%	**0.1686**	33.14%	**0.1364**	32.24%	0.1196	31.85%
	Redistricting	0.2564	34.73%	0.1697	**33.69%**	0.1384	32.74%	0.1218	30.67%
	Random	**0.2539**	**35.59%**	0.1694	32.65%	0.1371	**32.88%**	0.1202	31.61%
Spirals	PAL-ACS	**0.2816**	**38.82%**	**0.1927**	**58.07%**	**0.1397**	**66.61%**	**0.1139**	**66.81%**
	Inverse	0.2831	34.47%	0.2103	24.53%	0.1611	21.94%	0.1328	20.74%
	Redistricting	0.2861	30.61%	0.2165	21.17%	0.1735	15.96%	0.151	15.64%
	Random	0.2897	26.16%	0.2205	13.66%	0.1701	13.09%	0.1404	13.97%
Vehicle	PAL-ACS	0.5783	**34.45%**	0.4931	29.19%	0.4499	29.89%	0.4238	32.06%
	Inverse	0.5851	31.49%	0.5041	23.60%	0.4572	23.04%	0.4301	23.75%
	Redistricting	0.5783	33.62%	0.4981	26.00%	0.4536	28.84%	0.4290	26.78%
	Random	**0.5776**	34.32%	**0.4920**	**33.42%**	**0.4486**	**32.75%**	**0.4230**	**33.89%**
Vertebral	PAL-ACS	**0.3989**	34.00%	0.3696	31.55%	**0.3566**	**33.44%**	**0.3506**	**33.63%**
	Inverse	0.4088	30.53%	0.3764	27.61%	0.3625	27.97%	0.3536	28.99%
	Redistricting	0.4009	34.13%	0.3737	30.12%	0.363	28.57%	0.3557	27.25%
	Random	0.3993	**34.38%**	**0.3695**	**32.43%**	0.3578	32.51%	0.3522	31.85%
Yeast	PAL-ACS	0.4439	37.38%	**0.3909**	**29.99%**	**0.3716**	**29.59%**	**0.3606**	31.20%
	Inverse	0.4495	32.48%	0.3967	27.23%	0.3744	28.80%	0.3612	**31.64%**
	Redistricting	0.4444	35.07%	0.3958	26.99%	0.3762	27.39%	0.3659	23.67%
	Random	**0.4417**	**38.40%**	0.3931	28.48%	0.3731	27.95%	0.3617	28.21%

Table 1: Quantitative Comparison of ACS methods on all datasets. To show the learning process, the mean of errors has been calculated for different learning phases. Additionally, the ratio of won trials for each algorithm is shown. The winner is printed in bold.

4.2 Results and Discussion

To compare the algorithms, we provide learning curves in Fig. 5. These learning curves show the mean error and the variance of all algorithms with respect to the number of acquired instances. The best algorithm is the one that converges fastest to the lowest error.

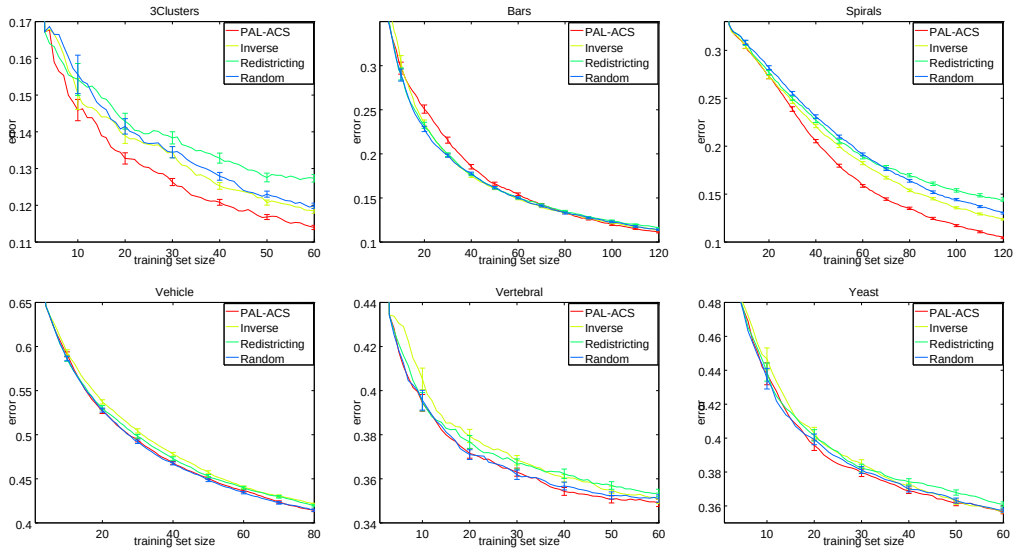


Figure 5: Learning Curves for each algorithm on every dataset. Each curve shows the mean error and standard deviation.

Additionally, we provide the quantitative values for the algorithms performances in Tab. 1. Here, we separated the learning process into four phases, in order to determine how fast algorithms get the structure of the learning problem. Each phase contains 25% of the learning steps. For each phase, we determine the mean accuracy for each algorithm on each dataset and calculate the ratio of won

trials. Note, that these ratios do not sum to one because some trials have multiple winners due to the aspects discussed at the end of Sec. 4.1.

First of all, the result from the plots and the table show that the current state-of-the-art approaches do not achieve considerable better results than random sampling, which justifies the development of our new ACS method. PAL-ACS is constantly better than both competing ACS methods with one exception. In the Bars dataset, our method performed better only towards the very end. This might be due to the non-Gaussian structure of the data as PAL-ACS internally uses Gaussian kernels to generate the pseudo instances. Comparing PAL-ACS to Random, we see that the superiority of our method depends on the structure of the data. The higher the differences in the complexity of the classes, the more beneficial is PAL-ACS.

PAL-ACSs high performance can be explained when looking at the sampled class distributions in Tab. 2. The results on 3Clusters, Spirals, and Bars in Tab. 2 show that PAL-ACS contributes a smaller sampling proportion to the easier class (1^{st} in 3Clusters and Spirals, 3^{rd} in Bars) than to the more difficult ones. Inverse and Redistricting show the same tendency, but to a much smaller extent, resulting in weaker performance in 3Clusters and Spirals. Although PAL-ACS showed mediocre results in Bars, it determined the easy class even in early phases.

Method	3Clusters	Bars	Spirals	Vehicle	Vertebral	Yeast
PAL-ACS	17,42,41	38,41,21	05,49,46	25,25,25,25	30,35,34	23,27,27,23
Inverse	29,35,36	35,36,30	28,36,36	27,27,24,23	38,36,25	28,28,22,23
Redist.	25,37,38	38,37,24	19,41,39	26,26,23,25	38,37,25	29,27,20,24
Random	33,34,33	33,33,34	33,34,33	25,25,25,25	33,34,33	25,25,25,25

Table 2: Final sampling proportions (for all classes) in percent.

Vehicle as a dataset with equally difficult classes demonstrates PAL-ACS's ability to detect this fact and converge to a uniform sampling proportion. Random performs slightly better on the Vehicle dataset as it has the advantage of assuming classes to be equally difficult per default, Inverse and Redistricting yield worse results by undersampling classes. On Vertebral and Yeast, we can see a clear sampling tendency in Tab. 2 which is also visible in the results of the learning curves in Fig. 5, resp. the performance table (Tab. 1).

Overall, PAL-ACS always identifies the difficult classes and samples accordingly. As a result, its performance is best (in cases some classes are more difficult than others) or equal with the best competitor Random (in cases all classes are equally difficult).

5 Conclusion

In this paper, we introduced a new approach for active class selection, called PAL-ACS (probabilistic active learning for active class selection). This method is based on the performance gain function proposed in [5] which was originally introduced for active learning. To apply this function, the ACS problem has been transformed in an active learning problem by generating pseudo instances.

The experimental evaluation shows our method's superiority on datasets where a non-uniform sampling improves the classifier's performance. On datasets with equally complex classes, our method identifies uniform sampling to be the best. Thus, in contrast to other active class selection methods, it performs comparably well with random sampling which is a uniform sampler per default.

In the future, we want to combine this approach with more sophisticated learning models and evaluate our algorithm on further datasets and real world BCI data. A further interesting topic is the comparison of our usefulness model with human information acquisition as mentioned in [11].

Acknowledgements

We thank the Psychoinformatics lab, esp. Michael Hanke and Alex Waite, from Magdeburg University for letting us use their cluster, our colleague Pawel Matuszyk, and the reviewers for their inspiring comments.

References

[1] Arthur Asuncion and David Newman. Uci machine learning repository, 2007.

[2] Josh Attenberg, Prem Melville, Foster Provost, and Maytal Saar-Tsechansky. Selective data acquisition for machine learning. In Balaji Krishnapuram, Shipeng Yu, and R. Bharat Rao, editors, *Cost-Sensitive Machine Learning*, chapter 5. CRC Press, 2011.

[3] Olivier Chapelle. Active learning for parzen window classifier. In *Int. Workshop on Artificial Intelligence and Statistics*, pages 49–56, 2005.

[4] Johannes Höhne, Elisa Holz, Pit Staiger-Sälzer, Klaus-Robert Müller, Andrea Kübler, and Michael Tangermann. Motor imagery for severely motor-impaired patients: Evidence for brain-computer interfacing as superior control solution. *PLoS ONE*, 9(8), 08 2014.

[5] Daniel Kottke, Georg Krempl, Dominik Lang, Johannes Teschner, and Myra Spiliopoulou. Multi-class probabilistic active learning. In Maria Fox, Gal Kaminka, Eyke Hüllermeier, and Paolo Bouquet, editors, *Proc. of the 22nd Europ. Conf. on Artificial Intelligence (ECAI2016), 2016*, Frontiers in Artificial Intelligence and Applications. IOS Press, 2016.

[6] Georg Krempl, Daniel Kottke, and Vincent Lemaire. Optimised probabilistic active learning (OPAL) for fast, non-myopic, cost-sensitive active classification. *Machine Learning*, Special Issue of ECML PKDD 2015, 2015.

[7] Georg Krempl, Daniel Kottke, and Myra Spiliopoulou. Probabilistic active learning: Towards combining versatility, optimality and efficiency. In Saso Dzeroski, Pance Panov, Dragi Kocev, and Ljupco Todorovski, editors, *Proc. of the 17th Int. Conf. on Discovery Science*, volume 8777 of *LNCS*, pages 168–179. Springer, 2014.

[8] Rachel Lomasky. *Active Acquisition of Informative Training Data*. PhD thesis, Tufts Univ., 2009.

[9] Rachel Lomasky, Carla E. Brodley, Matthew Aernecke, Sandra Bencic, and David Walt. Guiding class selection for an artificial nose. In *NIPS Workshop on Testing of Deployable Learning and Decision Systems*, 2006.

[10] Rachel Lomasky, Carla E. Brodley, Matthew Aernecke, David Walt, and Mark Friedl. Active class selection. In *Machine Learning: ECML 2007*, volume 4701 of *LNCS*, pages 640–647. Springer, 2007.

[11] Douglas B. Markant, Burr Settles, and Todd M. Gureckis. Self-directed learning favors local, rather than global, uncertainty. *Cognitive Science*, 40(1):100–120, 2016.

[12] Emmanuel Parzen. On Estimation of a Probability Density Function and Mode. *Annals of Mathematical Statistics*, 33(3):1065–1076, 1962.

[13] Nicholas Roy and Andrew McCallum. Toward optimal active learning through sampling estimation of error reduction. In *Int. Conf. on Machine Learning, ICML*, ICML '01, pages 441–448, San Francisco, CA, USA, 2001. Morgan Kaufmann.

[14] Burr Settles. Active learning literature survey. *Univ. of Wisconsin, Madison*, 2010.

[15] Burr Settles. *Active Learning*, volume 18 of *Synthesis Lectures on Artificial Intelligence and Machine Learning*. Morgan & Claypool, June 2012.

[16] Dongrui Wu, Brent J. Lance, and Thomas D. Parsons. Collaborative filtering for brain-computer interaction using transfer learning and active class selection. *PLoS ONE*, 8(2):e56624, 2013.

[17] Dongrui Wu and Thomas D. Parsons. Active class selection for arousal classification. In *Proc. of the 4th Int. Conf. on Affective Computing and Intelligent Interaction - Volume Part II*, ACII'11, pages 132–141. Springer, 2011.