Grant agreement no: FP7-600877

SPENCER:

Social situation-aware perception and action for cognitive robots

Project start: April 1, 2013

Duration: 3 years

DELIVERABLE 2.3

Unsupervised offline classification of dynamic objects

Due date: month 34 (January 2016)

Lead contractor organization: TUM

Dissemination Level: PUBLIC

Contents

Abstract

We present our progress on online learning from streams of data for 3D object classification. During the second half of the project, we developed several methods that address different aspects of this task. These methods were published in three conference papers, which we attached to this document. For any technical details, we refer to these papers, here we only summarize the developed algorithms.

1 Introduction

To automatically annotate observations from a given input data stream with semantic information, a robot must rely on some kind of human intervention. This means that a classification algorithm that is supposed to find moving objects such as cars and bicycles or – as is the case in SPENCER – carts and trolleys in an airport, must be given at least some known examples of these objects. Therefore, some kind of supervision is required to achieve this task. However, information required from a human supervisor is expensive, and it is important to perform the learning task with as few label queries as possible. Furthermore, for the scenario addressed in SPENCER, but also in many other robotic applications, learning must be done on the fly, i.e. the classifier must be able to update its internal representation quickly and without having to re-observe all previous data samples. Therefore, in task 2.2. we investigate online learning methods that require only little annotated data. One popular framework to achieve this is Active Learning, which we briefly describe in the next section. Then, in Sec. 3 we discuss the need to have classifiers that return a high uncertainty when they make wrong predictions (i.e. those that are less "overconfident") for active learning. In Sec. 4 we present a new method to address the problem of non-i.i.d data coming from streams. This appears particularly in online learning settings where current observations are highly correlated with those from previous frames. And in Sec. 5 we summarize our work on semi-supervised online learning for classification of moving objects. All findings presented in Sec. 3, 4, and 5 have been published in conference articles, and these articles are added in the appendix for a more detailed reference.

Note: The title of this deliverable is somehow misleading. The purpose of Task 2.2 is to investigate online (and not "offline") learning methods, and a completely unsupervised method was not the goal here, but rather semi-supervised and active learning techniques. In fact, as stated in the DOW (Task 2.2) "... the aim of this task is to reduce the necesity of human intervention as much as possible", and "A second goal of this task is to develop techniques that improve the learned models over time".

2 Active Learning

The main difference between standard passive learning and active learning is that, instead of strictly separating between a *training* and a *testing* phase, the active learner performs *loops* of training and testing, thereby incorporating the information flow obtained from the teacher (e.g. a human supervisor) into the loop. Fig. 1 shows a schematic flow chart of a generic active learning algorithm. We note that, while in general active learning can be used in many different contexts, we use it for object classification in this work.

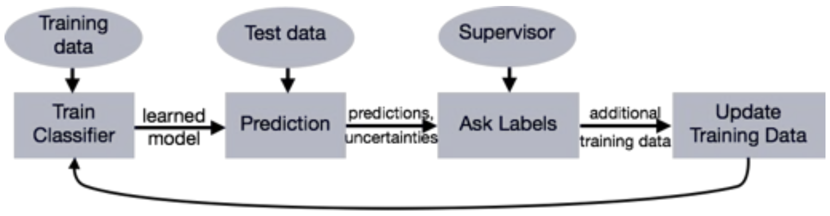


Figure 1: Active Learning flow chart. After an initial training step, the classifier is presented new test data and reports label predictions and confidence values (here: uncertainties). These are used to ask a human supervisor for new ground truth labels, which subsequently are added to the current training data. Then, the training process is repeated with the extended training data until a stopping criterion is met.

One important question in active learning is how to select the data samples for which semantic information, i.e. in our case class labels, are requested from the human supervisor. We refer to this as the *which-question problem*. The most used method to address it is uncertainty sampling, and we also use this in our implementation. Thus, we compute an uncertainty along with the prediction of a newly observed sample. Then, we use a *confidence threshold* ϑ_c and decide to ask for a ground truth label $\hat{y}$ for all those data samples which, in the current learning epoch, have been classified with a confidence lower than ϑ_c.

3 Confidence Boosting

One important requirement for a classfier used for Active Learning is that it is not overconfident, i.e. it should not give low uncertainty predictions when the prediction itself is actually wrong. Therefore, we developed the Confidence Boosting method (see attached paper, published at ICRA 2015). The main idea is to use the confidence (i.e. one minus the uncertainty) of a predicted sample to give those samples a higher weight that are wrong and certain. This way, in the boosting framework, the next weak learner will put more emphasis on those samples, thereby reducing the overconfidence of the entire (strong) classifier. Some results of this method are shown in Fig.2. As can be seen, Confidence Boosting is particularly useful for Active Learning applications.

4 Stream-Based Active Learning

In principle, there are two ways to do Active Learning: either the test set consists of observations that are collected beforehand, or it is a growing number of samples that are continously observed and added to the training data. In the first case we have a fixed *pool* of data, and the algorithm can pick good samples to query from this pool, which is usually very large. This is the most common application for Active Learning. However, in mobile robotics, and in particular for robots that are to learn semantics persistently, the second scenario known as *stream-based* Active Learning is much more relevant, because robots perceive streams of data, and they should be able to learn from it continuously. Therefore, for SPENCER we consider stream-based Active Learning.

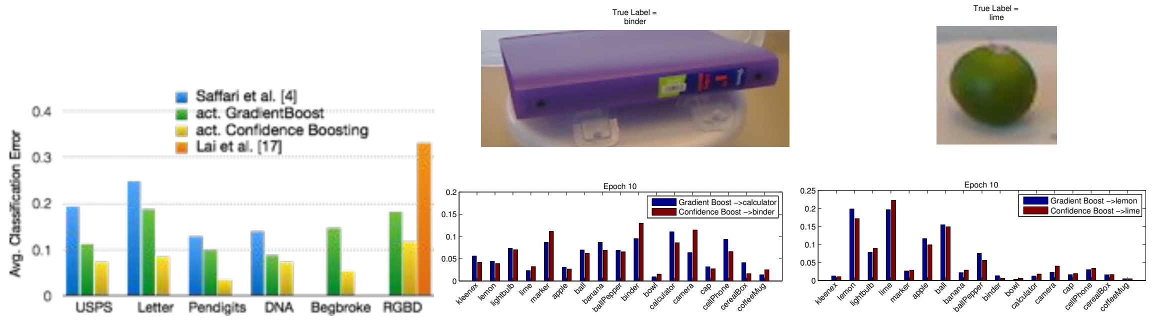


Figure 2: **Left:** Comparison of active learning with the results reported in the literature. Active Confidence Boosting always performs best. **Center and Right:** Qualitative results of our experiments. We show two different objects from the RGBD data set. Note that colour is not used for classification. After 10 rounds of active learning, Confidence Boosting (CB) returned the correct label, while standard gradient boosting did not. Note that CB even distinguishes the lime correctly from a lemon although there was no colour information used, i.e. even such small differences in shape can be detected with our approach.

4.1 Pool-based vs. stream-based learning

To analyse this difference further, we performed the following experiment. We considered a large, standard benchmark data set and applied two online classification methods: online Random Forests (ORF) and online multi-class Gradient Boost (OMCGB) with ORF as weak classifiers (for both methods see [1]). Then, we resampled the data in such a way that the occurence of samples in each class was distributed uniformly over the time line and applied again the online classification methods. Evaluation was done on a hold-out set not used for training, and we choose the KITTI data [2] for this experiment. The resulting learning curves are shown in Fig. 3 (left). As we can see, both online learning methods perform resonably well for the case of uniform distributions of class occurences. However, on the original data, where many objects of the same class can appear for some time period but for others there are almost no occurences, we have a significantly worse performance of the online learners. The reason for this behavior is that online Random Forests can not handle well data sets with unbalanced classes and with a non-uniform distribution of class occurences.

4.2 Improvement using Mondrian Forests

A novel algorithm that addresses the above problems is the *Mondrian Forest* by Lakshminarayanan *et al.* [3]. The major difference between a Mondrian tree and a standard decision tree is that Mondrian trees also store the *extent* of the data that corresponds to each node. While the decision tree uses splits that range over the entire *potential* range of the data, the splits of a Mondrian tree only cover the *actual* data range. This is achieved by keeping bounding box information for each sub tree. Note that the data itself is *not* stored in the tree, only the bounding boxes. As in random decision trees, splits are generated randomly, but samples are drawn from an exponential distribution whose rate parameter is proportional to the data extent of the sub tree.

Fig. 3 shows the resulting learning curve when using a Mondrian Forests instead of a Random

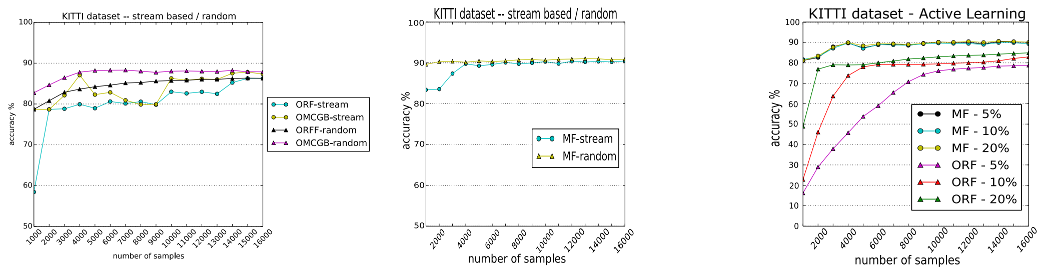


Figure 3: **Left:** Learning curves of two standard online learning methods: online Random Forests (ORF) and online multi-class Gradient Boost (OMCGB), both evaluated on the original and the re-sampled data ("stream" vs. "random"). As we see, the performance of both methods for the original data stream is significantly worse than for the resampled set. **Center:** Learning curves of the Mondrian forest for the same experiment. The MF classifier can deal much better with the data stream. **Right:** Classification accuracies for Active Learning using an MF and an ORF, where only 5%, 10% and 20% of the most uncertain data points are queried. Again, the MF clearly outperforms the ORF.

Forest for the re-sampled data set described above. We can see that the MF classifier increases its classification accuracy much faster than the ORF, even in the stream-based setting, and it also reaches a higher level (about 90% accuracy). We then tested the Active Learning scenario, where new label queries were generated after every 1,000 data samples. From these, we only used the most uncertain predictions for querying and re-training, and this fraction varied between 5% and 20%, i.e. from 50 to 200 samples per learning epoch. The result is shown in the right plot of Fig.3. The plot clearly shows that the MF can improve its classification accuracy even when trained only on a very small fraction of the data. Thus, the MF classifier both generates less queries and it can deal with the hard problem of learning from data streams. This is also reflected by the qualitative results shown in Fig. 4.

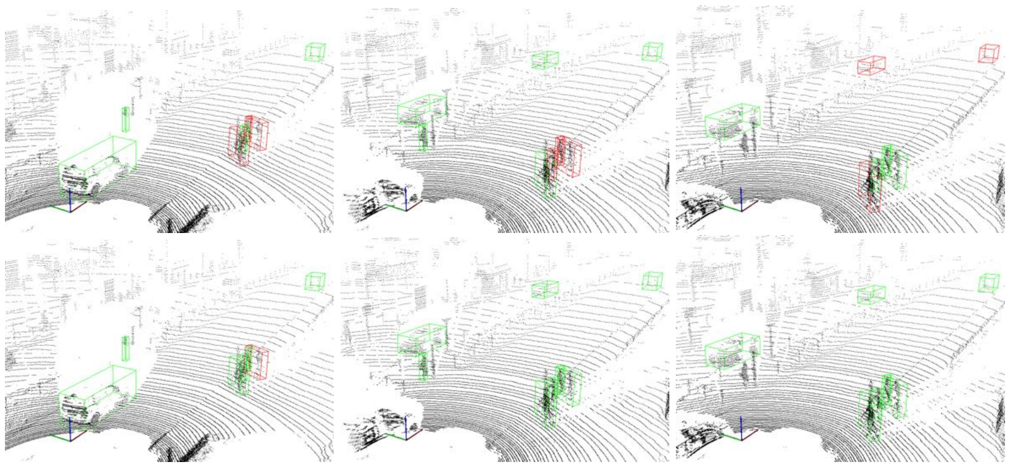


Figure 4: Some results on the KITTI data. From left to right, we show classification results on five consecutive data frames (point clouds) after training on a stream of 10000 samples. The upper row shows the result for ORF, the lower row the MF result. Most classifications are done correctly by the MF (green boxes), while the ORF has many false classifications (red boxes).

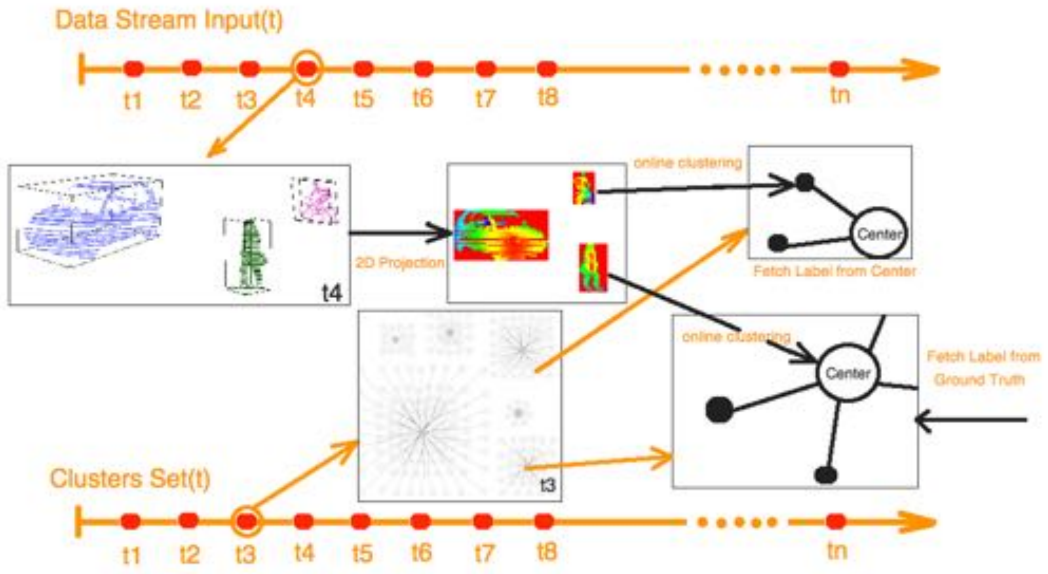


Figure 5: Overview of our online learning approach for 3D object classification from data streams. The upper time line represents the input from the data stream, the lower time line visualizes the online clustering method used. For new samples, there are two possibilities: either they are grouped into an existing cluster by adding them as "satellites" onto a "center" node, or they are themselves new center nodes. In the first case (here: the cyclist), no ground truth label is queried, but the label of the already existing center node is used. In the second case (here: the pedestrian), a new ground truth is queried.

5 Semi-Supervised Online Learning

As shown above, Active Learning is a useful method to reduce the number of required training samples, especially when the classification is done from data streams. However, the actual required number of samples can still be large, and therefore we aim at a further reduction in the number of hand-labeled training samples. To do this, we also investigate semi-supervised learning, i.e. methods that can deal with data sets that are partially labeled. The idea is to exploit the information about similarities between labelled and unlabelled samples in addition to the ground truth labels for learning. And, as above, we require that the method is online, i.e. that it is able to incrementally update the internal representation with newly observed samples.

In the attached publication at IROS 2015, we developed a semi-supervised online learning method that learns dynamic objects from a stream of 3D point clouds. The overview of this system is shown in Fig. 5. For the details of the algorithm, we refer to the attached paper.

The quantitative results are shown in Fig. 6. We evaluated the approach both on the KITTI data set mentioned earlier, as well as on the RGB-D data set by Lai *et al.* [4]. Form the plots we see that our method outperforms a standard combination of Online Star Clustering with Label Propagation, while the number of generated clusters is very similar.

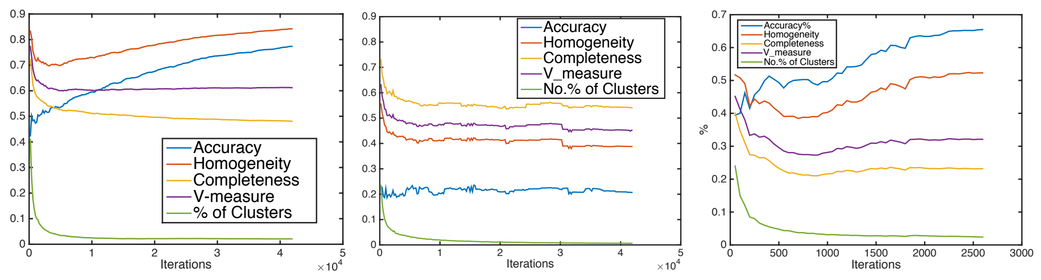


Figure 6: **Left and Center:** Evaluation on the RGB-D data set based on V-measure (violet), classification accuracy (blue) and number of clusters per vertex (green). **Left:** Our approach. **Center:** Combination of standard Online Star Clustering with Label Propagation (OSP+LP). Note that our approach significantly outperforms the OSC+LP method in terms of accuracy and V-measure, although there is no big difference in the number of clusters. **Right:** Result of our algorithm on the KITTI data set. The accuracy is worse than on the RGB-D set, but the input features are only based on depth values and not on color.

6 Discussion and Conclusions

In terms of the algorithmic development of efficient online learning methods for 3D object classification, with the additional requirement that the method only requires comparably little training data, we have made significant progress within SPENCER. This is mainly documented by the three publications, especially because for all three, the application of the developed methods to concrete perception problems in robotics was the main focus. That said, it still remains to run further tetst with these methods on the data sets produced by SPENCER at the airport of Amsterdam. Currently, the available data from preliminary tests can not be used due to the lack of other software components (e.g. data segmentation) and also because the focus during the integration week in Amsterdam was laid on other components of the system. We are however working on this and we will collect more data during the final deployment phase of the project. This will then allow us to show the usability of our methods for the SPENCER application, even though this might me after the end of the project.

References

[1] A. Saffari, M. Godec, T. Pock, C. Leistner, and H. Bischof, “Online multi-class lpboost,” in *CVPR*, 2010.

[2] A. Geiger, P. Lenz, and R. Urtasun, “Are we ready for autonomous driving? the KITTI vision benchmark suite,” in *Proc. of Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[3] B. Lakshminarayanan, D. M. Roy, and Y. W. Teh, “Mondrian Forests: Efficient Online Random Forests,” in *Advances in Neural Information Processing Systems (NIPS)*, 2014.

[4] K. Lai, L. Bo, X. Ren, and D. Fox, “A large-scale hierarchical multi-view rgb-d object dataset,” in *ICRA*, 2011.

Active Online Confidence Boosting for Efficient Object Classification

Dennis Mund Rudolph Triebel Daniel Cremers

***Abstract*— We present a novel efficient algorithm for object classification. Our method is based on the active learning framework, in which training and classification are performed in loops, and new ground truth labels are queried from the supervisor in each loop. Our underlying classifier is from the family of boosting methods, but in contrast to earlier methods, our Confidence Boosting particularly focusses on misclassified samples that have a high classification confidence associated. We show that weighting these samples more than others leads to a decrease of overconfidence, for which we give a formal definition. As a result, our classifier is better suited for active learning, leading to steeper learning curves and less required label queries. We show the benefits of our approach on standard data sets from machine learning and robotics.**

I. Introduction

Object classification is one of the most important tasks for a mobile robot, because for many kinds of interactions with the environment or with a human user, the robot needs semantic information about the environment. For that reason, research in this topic is performed by a large community, both within robotics and computer vision, and a number of good approaches have been presented in the past. The focus of these methods, however, differs slightly with the application: While in offline learning run time and memory efficiency are often less important, robotics is often concerned with online learning, because robots need to make fast decisions and update their internal representations with little computational effort. Furthermore, an important aim is to reduce the amount of required user interaction. In the case of object classification this means that the required amount of human-labeled training data should be small. Finally, for a mobile robot it is very advantageous to have a reliable measure of *confidence* along with the classification results, because often important and safety-critical decisions depend on them, and a restriction to very confident classifications can help to avoid accidents, for example in autonomous driving applications [1], [2].

In this paper, we present a novel learning method that addresses all these three issues. Our approach is based on an active learning framework, in which the human user is involved in the learning process, and learning is done in cycles of iterated training and inference. As we will show experimentally, this reduces the amount of required hand-labeled training data, and at the same time increases the classification performance. As an underlying classification method we use a novel online multi-class boosting algorithm. The motivation to choose this classifier stems from the very fast computation time used for training and inference, and from the online nature of the algorithm, i.e. it can update its internal representation and make predictions before having seen all input data. This is particularly useful in active learning, because it reduces the run time. However, as we will show, standard online boosting methods (e.g. Saffari *et al.* [4]), are not very useful for active learning, because they tend to associate wrong classifications with a too large confidence, i.e. they are often *overconfident*. This has a severe effect, because it prevents the classifier from finding misclassified samples and, as a result, reduces the chance to obtain ground truth labels from the human for re-learning these erroneously classified samples. An illustrative example is given in Fig. 1. Here, the input image shows a light bulb, but the available information is too low to classify this object correctly. Both standard gradient boosting and our Confidence Boosting method report a wrong class label. However, after 10 rounds of active learning (on a data set that is different from this test data), Confidence Boosting is able to classify the object correctly, while gradient boosting is not. In our experiments, we will show that this is due to the reduced overconfidence of our classification algorithm.

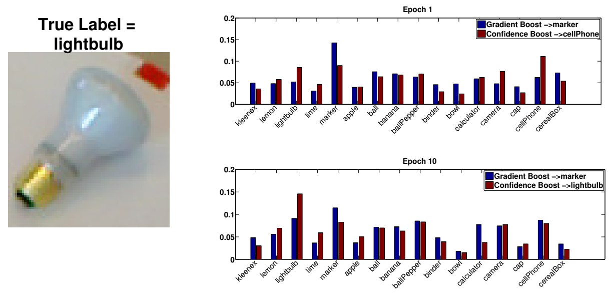


Fig. 1: Object classification with active online Confidence Boosting: The image on the left was recorded with an RGB-D sensor [3]. On the right, we show class label predictions obtained from online gradient boosting [4] and our online Confidence Boosting. The upper row shows results after one learning round, the lower row depicts results after 10 rounds of active learning. As we can see, both classifiers first return a wrong class label (see arrows in legend), but Confidence Boosting can recover from the error and finally give a correct label. In this paper, we show that this is because Confidence Boosting is less overconfident.

A. Related Work

The existing literature on object classification is too large to be mentioned exhaustively here. Therefore we only name some of the most recent achievements. These include deep belief nets [5], convolutional deep belief networks (CDBN) [6] and fully connected Conditional Random Fields (CRFs) [7]. Despite their impressive results, these classifiers focus

All authors are with Computer Vision Group, Department of Computer Science, Technische Universität München, Germany {dennis.mund,rudolph.triebel,daniel.cremers}@in.tum.de

only on the classification rate, whereas we are are also interested in the overconfidence of a classifier. Furthermore, sparse coding techniques have become popular, and in particular the Hierarchical Matching Pursuit (HMP) algorithm [3], which we also use to compute descriptors, however with a classifier that is superior in performance compared to the linear support vector machine (SVM) used there.

Our approach is formulated as an active learning method. This research area is experiencing a growing interest in the area of computer vision and robotics. For example, Kapoor *et al.* [8] use active learning for object categorization using a Gaussian Process classifier (GPC) where labels are queried for data points that are classified with high uncertainty. Triebel *et al.* [9] use a sparse version of the GPC, the Informative Vector Machine (IVM) to actively learn traffic lights in urban traffic images. In computer vision, Vezhnevets *et al.* [10], as well as Wang *et al.* [11] use active learning for interactive image segmentation. In contrast to all these methods, we propose to use a boosting method as an underlying classifier, because it is more efficient in terms of training and evaluation time. In that context, a work that is very closely related to ours is that of Saffari *et al.* [4], which proposes an efficient multi-class online boosting algorithm. We extend that approach using a similar idea as presented in [12], with the difference that here we use it for online boosting and with more theoretical justifications. Concretely, we introduce a formal definition of over- and underconfidence of a classifier on a given data set. This is related to the intuitive notion of *introspection* introduced by Grimmett *et al.* [1], however with the difference that our formulation can explicitly quantify the inherent trade-off between a high number of detected misclassifications and a high number of correct classifications that are not further used for training.

II. Active Learning

The main difference between standard passive learning and active learning is that, instead of strictly separating between a *training* and a *testing* phase, the active learner performs *loops* of training and testing, thereby incorporating the information flow obtained from the teacher (e.g. a human supervisor) into the loop. Fig. 2 shows a schematic flow chart of a generic active learning algorithm. We note that, while in general active learning can be used in many different contexts, we will use it for object classification in this work. In the following, we will describe the indivudal components of our framework in more detail.

A. Components of Active Learning

Similar to standard passive learning methods, in active learning we start with an initial training set $(\mathcal{X}_0, \mathcal{Y}_0)$, where $\mathcal{X}_0 = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$ are the N feature vectors with d dimensions, i.e. $\mathbf{x}_i \in \mathbb{R}^d$, and $\mathcal{Y}_0 = \{y_1, \dots, y_N\}$ are the corresponding class labels, i.e. $y_i \in \{1, \dots, C\}$. In this paper, we assume the number of classes C as given, but usually larger than 2. We note that, in contrast to passive learning, active learners usually can deal with much smaller initial

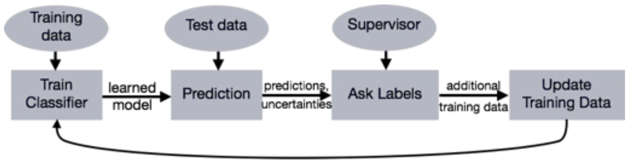


Fig. 2: Active Learning flow chart. After an initial training step, the classifier is presented new test data and reports label predictions and confidence values (here: uncertainties). These are used to ask a human supervisor for new ground truth labels, which subsequently are added to the current training data. Then, the training process is repeated with the extended training data until a stopping criterion is met.

training data sets, which is one major advantage of active learning methods. The first step is then to train a classifier with the initial training set, which we model as a function $f : \mathbb{R}^d \to \mathbb{R}^C$, i.e. each input feature vector $\mathbf{x}$ is mapped to a prediction vector $\mathbf{p} \in \mathbb{R}^C$. We will give more details on this in Sec. III. Next, a set of K data samples $\{\mathbf{x}_1^*, \dots, \mathbf{x}_K^*\}$ is drawn from the test data set $\mathcal{X}^*$ and classified using f. Here, the nature of $\mathcal{X}^*$ defines the type and complexity of the active learning problem: if $\mathcal{X}^*$ is given beforehand and its size does not change during learning, then we are concerned with *pool-based* active learning. If, however $\mathcal{X}^*$ is a potentially infinite stream of data, then we have *stream-based* active learning, which is significantly harder. In our work, we consider the pool-based variant, although we aim to extend the approach to stream-based learning in the future.

The result of the classification of $\{\mathbf{x}_1^*, \dots, \mathbf{x}_K^*\}$ are the *label predictions* $y_1^*, \dots, y_K^*$, where

$$y_k^* = \arg\max_c (p_1, \dots, p_C), \qquad \mathbf{p} = f(\mathbf{x}_k^*). \tag{1}$$

In addition to the label predictions, the prediction vectors themselves also play an important role, because they can be used to distinguish test points, where the classifier has a high chance of correct classification from those where the classification is likely to be incorrect. We refer to this as the *confidence* of the classifier. This can be computed based on the *uncertainty* of the classification, and we will give more details below. Now, the key element of active learning is the ability of the learner to *query* new class labels from the human supervisor. This is usually done by selecting those test points $\mathbf{x}_i^*$, for which the classifier has a low confidence and asking a ground truth label $\hat{y}_i$ for them. Here, we note that the number K of new test samples considered can be used to bound the required human effort by querying only the least confident ones. For $K = 1$ this would result in a query for every sample, but for larger K the benefit becomes obvious. Then, after the label query, the new data-label pairs $(\mathbf{x}_i^*, \hat{y}_i)$ are added to the current training data $(\mathcal{X}_{j-1}, \mathcal{Y}_{j-1})$, where j is the index of the current learning round, and the learning process starts again until an appropriate stopping criterion is reached. In our implementation, we use a fixed number of learning rounds.

B. The Which-Question Problem

One important question in active learning is how to select the data samples for which semantic information, i.e. in our case class labels, are requested from the human supervisor. We refer to this as the *which-question problem*. In the survey of Settles [13] the following *query strategies* are summarized to address this problem: uncertainty sampling, query-by-committee, expected model change, expected error reduction, variance reduction, and density weighting. Among these, the most used method is the uncertainty sampling, and we also use it in our implementation. Assume that the entries of the prediction vector $\mathbf{p}$ returned by f for a test sample $\mathbf{x}^*$ sum up to 1, i.e. $\sum_{i=1}^{C} p_i = 1$. Then, each entry p_i in $\mathbf{p}$ can be interpreted as the probability that $\mathbf{x}^*$ has label i. From this, there are two common ways to compute uncertainty:

$$h_e(\mathbf{p}) := -\sum_{i=1}^{C} p_i \log_C(p_i) \qquad h_b(\mathbf{p}) := p_{i_2}/p_{i_1}, \qquad (2)$$

where i_1 and i_2 are the indices of the largest and second largest values of $\mathbf{p}$, respectively. The first measure is the *normalized entropy*, where the base of the logarithm is C so that h_e always ranges between 0 and 1, and the second is a variant of the *best-vs-second-best* (BVSB) method. Both measures are usually very well suited for active learning, while BVSB tends to perform slightly better for multi-class classification tasks such as ours. With these definitions of uncertainty h, we define the classification confidence as $1-h$, and we will use both terms in the following.

Now, to address the which-question problem, the standard uncertainty sampling approach uses a *confidence threshold* ϑ_c and decides to ask for a ground truth label $\hat{y}$ for all those data samples which, in the current learning epoch, have been classified with a confidence lower than ϑ_c. This directly raises two questions: What is a good choice for ϑ_c? And how do we know that the classifier gives meaningful uncertainty estimates so that uncertainty sampling actually makes sense? While the first question will be answered in Sec. III-C, the second one will be addressed next.

C. Under- and Overconfidence

A crucial point with the uncertainty estimates obtained from the classification is the question, how much one can rely on these estimates. Formally, our aim is to have a high correlation between prediction uncertainty and incorrectness of the classification (a similar idea was used by Zhang et al. [14]). In [12], this correlation was measured using the point-biserial correlation coefficient. However, this turns out to be too restrictive and only applicable in cases where the number of correctly and incorrectly classified samples is roughly balanced. Therefore, we take a different approach. For a test set $\mathcal{X}^*$ of size K we define two functions u and o as follows:

$$u(f, \mathcal{X}^*, \hat{\mathcal{Y}}) := \frac{1}{K_c} \sum_{\mathbf{x}^* \in \mathcal{X}^*} I(y^* = \hat{y}) h(f(\mathbf{x}^*)) \qquad (3)$$

$$o(f, \mathcal{X}^*, \hat{\mathcal{Y}}) := \frac{1}{K_f} \sum_{\mathbf{x}^* \in \mathcal{X}^*} I(y^* \neq \hat{y})(1 - h(f(\mathbf{x}^*))), \qquad (4)$$

where I is the indicator function, $\hat{\mathcal{Y}}$ are the ground truth labels, and K_f and K_c are the number of incorrectly and correctly classified test samples, i.e. $K_f + K_c = K$. Thus, u is the average *uncertainty* of the *correct* classified samples and o is the average *confidence* of the *incorrectly* classified samples. We will denote u as the *underconfidence* and o as the *overconfidence* of the classifier. Intuitively, if all incorrect classified samples have the maximum confidence of 1 assigned, then the overconfidence reaches its maximum value of 1. This is the worst case for active learning, because the classifier is unable to give an indication that its predicted class label is wrong. As a consequence, the algorithm will never ask ground truth labels for the incorrectly classified samples, and they can not be used for re-training. Thus, the classification can not be improved.

Another extreme case is that of maximal *underconfidence* u. Here, all uncertainty values h for correctly classified samples are 1, i.e. the classifier is always fully uncertain, although the corresponding predictions are correct. The problem with this case is that active learning will be very inefficient, because very often the algorithm queries ground truth labels for samples that are already correctly classified. It is important to note that underconfidence and overconfidence are in this definition completely independent quantities. In particular, a classifier can be under- and overconfident at the same time, namely when it is uncertain on the correct predictions and certain on the wrong ones.

Despite the problems with inefficiency caused by under-confident classifiers, we will focus more on the task to avoid overconfidence, as this has the more severe effect on active learning. However, the problem here is that we can not explicitly minimize o using a closed-form expression. Also, we have to make sure that the overconfidence is reduced *simultaneously* with the reduction of the training error. To do this, we propose to extend a standard online multi-class boosting algorithm in such a way that it also takes classification confidences into account. This will be described next.

III. Online Confidence Boosting

In principle, there are two different ways to achieve classification results with little overconfidence: either we use a classifier that is already known to be less overconfident than others, or we modify an existing algorithm so that it is less overconfident. If we follow the first idea, then a good choice for a classifier is the Gaussian Process classifier (GPC), as was shown earlier [1], [15], because due to its capability to marginalize over a range of potential models, its uncertainty estimates are more reliable because they correlate more with actual misclassifications (see [1] for more details). One major problem however, is its huge demand in run time and memory. Even though there are sparse and more efficient variants such as the Informative Vector Machine (IVM) [16], the method is still hardly applicable for typical data sets in mobile robotics. Furthermore, as we are investigating active learning here, we have even stricter requirements on the run time, because the user is involved in the learning process, and learning should be done during operation of the robot.

Algorithm 1: Online Multi-class Gradient Boost [4]

```
Data: training data $(\mathcal{X}, \mathbf{y})$ with $C$ classes
Input: number of weak learners $M$, loss function $\ell$,
       agreement function $a$
Output: weak learners $f_1, \ldots, f_M$
1  Initialize($f_1, \ldots, f_M$)
2  for $n = 1, \ldots, N$ do
3      $w_n \leftarrow 1$
4      $g_n \leftarrow 0$
5      for $m = 1, \ldots, M$ do
6          $f_m \leftarrow$ UpdateWeakLearner($f_m, \mathbf{x}_n, y_n, w_n$)
7          $\mathbf{p}_{nm} \leftarrow f_m(\mathbf{x}_n)$
8          $\alpha_{nm} \leftarrow a(\mathbf{p}_{nm}, y_n)$
9          $g_n \leftarrow g_n + \alpha_{nm}$
10         $w_n \leftarrow -\nabla\ell(g_n)$
```

Therefore, we decided to use a classification framework that is known to be efficient and effective, and that can be modified so that it is less overconfident. A recently developed method with these requirements is the Online Multi-Class Gradient Boost (OMCGB) algorithm of Saffari *et al.* [4] (see Algorithm 1), which we describe next.

A. Online Multi-Class Gradient Boost

In addition to the training data, the OMCGB algorithm requires three different parameters as input: a fixed number M of weak learners, a loss function $\ell : \mathbb{R} \to \mathbb{R}$, and an *agreement function* $a : \mathbb{R}^C \times \mathbb{N} \to \mathbb{R}$, which quantifies the amount of agreement between a class label prediction $f(\mathbf{x}_n)$ and the corresponding ground truth label y_n. After initialization of the weak classifiers, the algorithm loops over all training data points and updates all weak classifiers for every new training sample $(\mathbf{x}_n, y_n)$. This online behaviour of the algorithm is very attractive for our active learning framework, because it avoids a recomputation of the underlying representation whenever a new ground truth label is queried from the user and added to the existing training set. As in offline boosting methods, every training sample $\mathbf{x}_n$ has an assigned weight w_n, which is first initialized to 1. Then, every weak classifier is first updated with the new sample, its weight w_n and its ground truth label y_n. Note that the weak learner itself also must be an online algorithm, because otherwise the overall boosting method would not be online. In our experiments we used online random forests as weak classifiers.

The next step (line 7) is to obtain a label prediction $\mathbf{p}_{nm}$ for the new training sample. Then, the agreement with the ground truth label is computed. In standard OMCGB, this is defined as

$$a_g(\mathbf{p}_{nm}, y_n) = p_{nm}^{(y_n)} - 1/C, \tag{5}$$

i.e. it is directly related to the prediction for class y_n, here denoted as an upper index into the prediction vector $\mathbf{p}_{nm}$. The resulting agreement α_{nm} is then accumulated, and a new weight w_n is computed for the sample from the negative gradient of the loss function of the sum of agreements. In [4], two different loss functions are investigated, but with little performance difference, so we decided to use the standard exponential loss $\ell(g) = \exp(-g)$ known from AdaBoost. Concretely, the computation in line 10 results in higher weights for samples that disagree with the ground truth and lower weights for those that do agree.

B. Extension to Confidence Boosting

As can be seen from Eq. (5), the agreement a_g used by standard gradient boost is only related to the prediction itself, but not to the confidence of the prediction. To build a classifier that takes both prediction and confidence into account, we propose to use this agreement function:

$$a_c(\mathbf{p}_{nm}, y_n) = (-1)^{\xi}\left(1 - \frac{h(\mathbf{p}_{nm})}{(C-1)^{\xi}}\right), \tag{6}$$

$$\text{where } \xi = I(\arg\max_i \mathbf{p}_{nm}^{(i)} \neq y_n). \tag{7}$$

This means, that we measure agreement by the amount of confidence, which is equal to one minus uncertainty. In case of a correct classification, i.e. when $\xi = 0$, the agreement simply amounts to the confidence of the current weak classifier f_m. However, if the classification is incorrect, we actually have a disagreement, and we express this with the – slightly modified – *negative* confidence. Our modification is the term $(C-1)$, by which we divide the uncertainty. This has empirically shown to improve the classification results substantially. To summarize, our agreement function is high if the classification is correct and certain, and it is low if we have an incorrect, but certain classification. Also note that if the uncertainty is zero, i.e. when we completely trust the classification, then the agreement is 1 for correct and -1 for incorrectly classified samples. Thus, in this case, our agreement function is even simpler than the original one given in Eq. (5).

C. Adaptive Thresholding

As mentioned in Sec. II-B, active learning with uncertainty sampling requires to specify the confidence threshold ϑ_c to decide for which samples a ground truth label should be queried. Usually, this threshold is a fixed parameter of the algorithm that does not change during the learning process. However, apart from the fact that it is in general difficult to find a good value for ϑ_c, having a fixed value often leads to a poor performance either in terms of efficiency or in terms of classification rate. The reason is that there is an inherent trade-off in the choice of ϑ_c. If it is small, then the number of label queries is low, thus increasing efficiency in the next learning round, but with a potentially lower classification rate. In contrast, if ϑ_c is large we have a higher chance of finding those samples, for which the classifier was wrong, which is good to correct for misclassifications, but it also increases the risk of re-learning already correctly classified samples. In addition to this, ϑ_c should also be chosen according to the level of over- and underconfidence of the classifier. For example, if the classifier tends to be overconfident, then a higher value of ϑ_c should be used to increase the chances to find misclassifications. To address this issue, in our implementation we use an adaptive method: In every training epoch we compute confidence histograms for correctly and incorrectly classified samples from the previous epochs. Then we start a search at $\vartheta_c = 0$ with a positive step size until the fraction of false samples with a confidence below ϑ_c equals the fraction of correct samples with a confidence above ϑ_c. As we will see later in the experiments, this method provides a good compromise, and it uses a similar idea than the equal-error-rate in a precision-recall graph.

IV. Experimental Results

To evaluate our algorithm, we performed three different experiments on six different data sets. The first experiment investigates how much Confidence Boosting actually reduces the overconfidence in the class label predictions. In the second, we analyze the impact of Confidence Boosting within the active learning framework. And in the last experiment, we compare the learning curve and the run time of Confidence Boosting with those of a Gaussian Process Classifier, which is known to perform particularly well in active learning. All data sets and experiments are described in more detail next.

A. Data Sets

We used four data sets from the UCI machine learning repository, and two sets from robotics. The UCI data sets are ‘USPS’, ‘Pendigits’, ‘Letter’, and ‘DNA’. We used these because they were also used for evaluation by Saffari *et al.* [4], and our aim is to compare Confidence Boosting with gradient boosting. The robotics data sets we used were an RGB-D set provided by Lai *et al.* [17], and the 3D point cloud data from Paul *et al.* [15]. From the first one, for which we use the identifier ‘RGBD’, we extracted 89 pre-segmented objects of 17 object classes, resulting in a total of 58372 RGB-D images. Then, we computed Hierarchical Matching Pursuit (HMP) descriptors [3] on the depth channel. The dictionary needed for the HMP features was learned on 5 classes out of 17, mainly for memory reasons. Then, the data was split into a training set of 90% of the data and an evaluation set of the remaining 10 %. The other robotics data set, which we denote as ’Begbroke’, consists of 3D point clouds from a car park with 6 classes. The data was segmented automatically, and features were computed for each segment (see [15]). In total, there were 1496 segments, out of which we took 1000 for training and the rest for evaluation. We used this data to be able to compare with the multi-class GP classifier used in [15], both in terms of run-time and classification performance.

B. Reducing Overconfidence

To measure how much Confidence Boosting actually reduces overconfidence, we computed histograms over the confidences for correctly and incorrectly classified samples, both for gradient boosting and for Confidence Boosting. Fig. 3 shows the resulting histograms for gradient boosting (GB) and Confidence Boosting (CB) on the ‘DNA’ data set. The plots on the left show the confidence histograms for the correct classified samples, the right ones for the incorrect samples (red bars depict either over- or under-confident regions). As we can see, Confidence Boosting tends to shift both histograms to the left, which means that in general classification is more uncertain. This implies that more false classifications are uncertain as well. Thus, increasing the uncertainty in general reduces overconfidence, but it also increases underconfidence. However, as we can see, Confidence Boosting shofts the histograms for the false samples more than the correct ones. A quantitative result is shown in Table I. Note that, e.g. for the ‘DNA’ data set, the overconfidence is lower both for passive and for active learning when using Confidence Boosting. Although this does not hold for all data sets (see, e.g. ‘USPS’), one can say that in general the tendency is that Confidence Boosting reduces the overconfidence.

To visualize the trade-off between over- and underconfidence, we use a plot type similar to a precision-recall curve. On the x-axis, we plot for a given number of different confidence thresholds $\theta_1, \theta_2, \ldots$ the fraction of false classified samples that have a confidence below the thresholds. On the y-axis, we plot for the same thresholds the fraction of correct classified samples for which the classification was more confident than θ_i. Ideally, this curve stays close towards the upper right corner of the plot. Fig. 4 shows an example of such a plot for the ‘Pendigits’ data set, both for gradient boosting and for Confidence Boosting. We see that Confidence Boosting gives the opportunity to

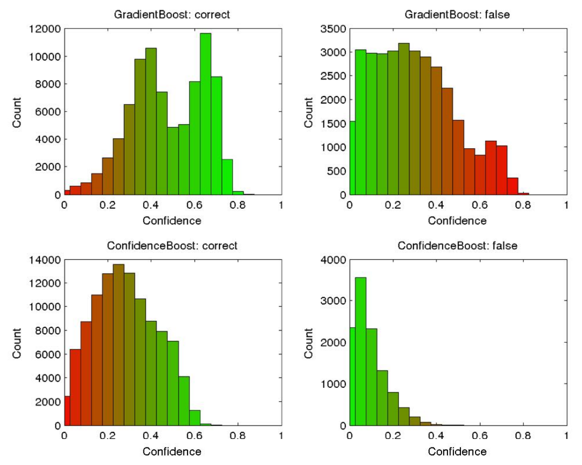
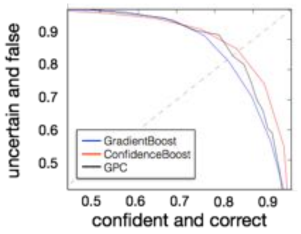


Fig. 3: Confidence histograms of gradient boosting and Confidence Boosting on the ‘DNA’ dataset. The left plots show the histograms for the correctly classified samples, the right ones show the histograms for the incorrect samples. Confidence Boosting shifts all histograms to the left, resulting in an overall decreased confidence. Note however that the false samples are shifted even further, which leads to a lower overconfidence.

	RGBD	Begbroke	USPS	Letter	Pendigits	DNA
pGB	0.1326	0.3106	0.1978	0.1854	0.2804	0.1475
aGB	0.1064	0.2391	0.1653	0.1395	0.2192	0.1258
pCB	0.0981	0.1715	0.1624	0.1416	0.1682	0.0895
aCB	0.0960	0.1629	0.1515	0.1338	0.1684	0.0871

TABLE I: Overconfidence averaged over 100 runs. We compare passive and active Gradient Boost (pGB and aGB) with passive and active Confidence Boost (pCP and aCB).



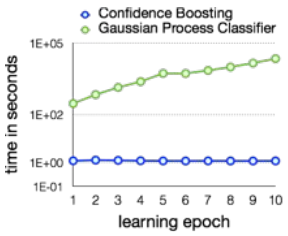


Fig. 4: **Left:** Trade-off curves for gradient boosting, Confidence Boosting, and the GPC on the 'Pendigits' data set. Higher curves correspond to less overconfident classifiers, curves that are further to the right represent a lower underconfidence. We see that Confidence Boosting generally improves over- and underconfidence compared to gradient boosting, and it is even less underconfident than the GPC. **Right:** Average run times of Confidence Boosting (blue) and the GPC (red) for each epoch (note the log scale).

	RGBD	Begbroke	USPS	Letter	Pendigits	DNA
pGB	0.2704	0.2668	0.1536	0.2628	0.1589	0.1410
aGB	0.1824	0.1463	0.1123	0.1876	0.0983	0.0867
pCB	0.1248	0.0568	0.0898	0.1161	0.0559	0.0811
aCB	0.1165	0.0517	0.0726	0.0840	0.0341	0.0724

TABLE II: Average classification errors over 100 runs.

'detect' more false classifications while at the same time not loosing too many correct classifications. The plot also shows how to choose a good confidence threshold ϑ_c for active learning. While our adaptive method chooses the one where the false and the correct classification rates are equal, one could focus more on efficiency by not loosing many correct classifications (towards the right part of the graph) or on classification quality by including more false classified samples into the training set (towards the upper part).

C. Impact of Confidence Boosting for Active Learning

To quantify the effect of Confidence Boosting in the active learning framework, we ran active learning on all 6 data sets, once with standard gradient boosting and once with Confidence Boosting. We repeated this 100 times with the training sets randomly shuffled to obtain results that are independent on the data ordering. The mean classification errors are given in Table II. Apart from the fact that active learning performs better on all data sets than passive learning, where training data was selected randomly and not based on its confidence, we see that Confidence Boosting results in lower classification errors. In particular, Confidence Boosting with active learning performs best on all data sets.

The progress of the learning process for 'Pendigits' is depicted in Fig. 5 (left). Here, the means and standard deviations of the classification error are shown as a function of the generated label queries. We see that Confidence Boosting leads to better results and requires less label queries at the same time. For comparison we also show results on passive learning where the training data was as large as the one for active learning was at the end. Thus, even though passive learning uses the same amount of data, the active learner is better after some learning epochs. Fig. 5 (center) explicitly shows the number of label queries per epoch for three different data sets. Again, Confidence Boosting produces less queries than Gradient Boosting. We relate this to the fact that

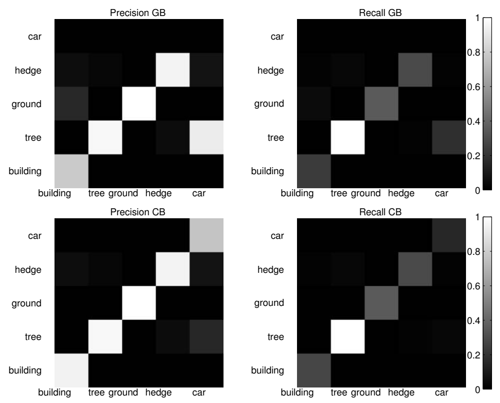


Fig. 7: Normalized confusion matrices for active gradient boosting (GB) and Confidence Boosting (CB) on the 'Begbroke' data set (see text for details).

Confidence Boosting results in a higher correlation between uncertain and incorrect classified samples, which enables the classifier to find more incorrect samples for re-training while at the same time not loosing many correct classified samples. The right part of Fig. 5 shows a comparison of our results from Table II with those reported in [4] and [17]. Note that [4] perform learning on the same data for a number of rounds, which can not be compared to our active learning epochs. Therefore we only compare to the first round reported in [4]. Some qualitative results of our approach are shown in Fig. 6.

D. Comparison to the GPC

In [1], [15] it was shown that a GPC tends to be much less overconfident than other classifiers such as a Support Vector Machine (SVM) or LogitBoost. However, here we show that Confidence Boosting mitigates this issue at least for boosting methods, and we relate that not only to the fact that the confidence is used to compute the sample weights, but also to the choice of the weak classifiers, namely random forests. We argue that with this combination we have a very efficient classification framework that also reduces overconfidence in a way that it comes close to that of a GPC. To back-up this claim quantitatively, we ran a multi-class GPC on the same data sets using an own implementation based on [18]. Unfortunately, for most data sets GPC training could not be done efficiently, therefore we only report results on the 'DNA' data set. Fig. 4 (left) shows that the GPC is only slightly less overconfident than Confidence Boosting. The classification error of the GPC for 'DNA' after 10 epochs was 0.0552, which is slightly better than active Confidence Boosting (see Table II), but at the cost of a much higher run time (see Fig. 4, right).

We also compare our results on the 'Begbroke' data with those reported in [15]. For that, we computed the same normalized confusion matrices as in [15], although without the underrepresented 'Background' class (see Fig. 7). The figure shows that our active Confidence Boosting method reaches a classification performance that is almost as good as that of the GPC, but with a much faster computation time.

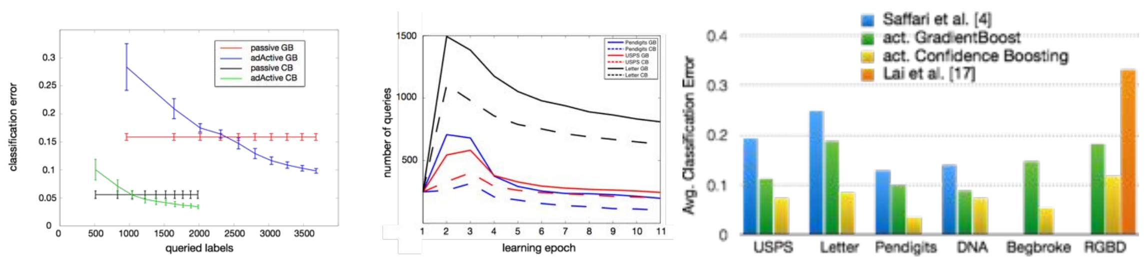


Fig. 5: **Left:** Learning curves for the 'Pendigits' data set. The x-axis shows the number of label queries generated by active learning. Both Gradient Boosting and Confidence Boosting improve with more samples, but Confidence Boosting reaches lower errors and requires less label queries. For comparison, the passive counter parts are shown where the same number of data points was used for training as the active learner has at the end. **Center:** Number of new label queries used per epoch for 'Pendigits', 'USPS' and 'Letter'. All curves start with a pool of 250 samples. We see that in each epoch less additional queries are needed than in the previous one and that Confidence Boosting needs less label queries than gradient boosting. **Right:** Comparison of active learning with the results reported in the literature. Active Confidence Boosting always performs best.

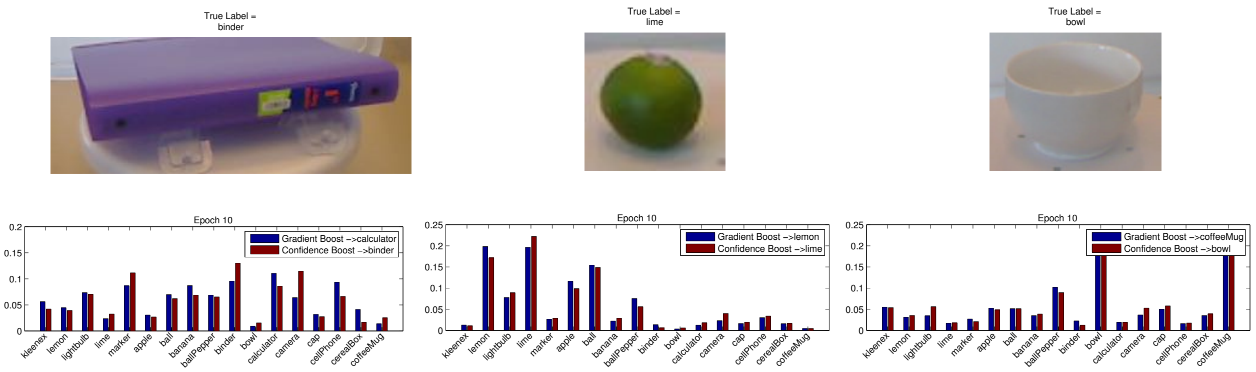


Fig. 6: Qualitative results of our experiments. We show three different objects from the RGBD data set, for which we computed HMP descriptors on the depth information, i.e. colour is not used for classification. After 10 rounds of active learning, Confidence Boosting (CB) returned the correct label, while gradient boosting did not. Note that CB even distinguishes the lime correctly from a lemon although there was no colour information used, i.e. even such small differences in shape can be detected with our approach.

V. Conclusions

Object classification still remains a difficult problem. However, active learning seems to be a very useful technique to tackle this problem, even though it raises the questions of training efficiency and low overconfidence of the classifier. While the latter can be handeled well using a Gaussian Process classifier, this same method is often too slow for online applications. In contrast, our proposed extension of online multi-class gradient boosting is at the same time very efficient and reduces overconfidence over standard boosting, coming close to that of the GPC. The result is an efficient and high performing active learning method, which gives good results even on challenging state-of-the-art data in robotics.

References

[1] H. Grimmett, R. Paul, R. Triebel, and I. Posner, "Knowing when we don't know: Introspective classification for mission-critical decision making," in *ICRA*, 2013.

[2] P. Lamon, C. Stachniss, R. Triebel, P. Pfaff, C. Plagemann, G. Grisetti, S. Kolski, W. Burgard, and R. Siegwart, "Mapping with an autonomous car," in *IEEE/RSJ IROS Workshop: Safe Navigation in Open and Dynamic Environments*, 2006.

[3] L. Bo, X. Ren, and D. Fox, "Hierarchical matching pursuit for image classification: Architecture and fast algorithms," in *Advances in neural information processing systems*, 2011, pp. 2115–2123.

[4] A. Saffari, M. Godec, T. Pock, C. Leistner, and H. Bischof, "Online multi-class lpboost," in *CVPR*, 2010.

[5] G. Hinton, S. Osindero, and Y. Teh, "A fast learning algorithm for deep belief nets," *Neural Computation*, vol. 18, no. 7, 2006.

[6] H. Lee, R. Grosse, R. Ranganath, and A. Ng, "Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations," in *International Conference on Machine Learning (ICML)*, 2009.

[7] P. Krähenbühl and V. Koltun, "Efficient inference in fully connected crfs with gaussian edge potentials," in *Conf. on Neural Information Processins Systems (NIPS)*, 2011.

[8] A. Kapoor, K. Grauman, R. Urtasun, and T. Darrell, "Gaussian processes for object categorization," *Intern. Journal of Computer Vision*, vol. 88, no. 2, pp. 169–188, 2010.

[9] R. Triebel, H. Grimmett, R. Paul, and I. Posner, "Driven learning for driving: How introspection improves semantic mapping," in *Proc of Intern. Symposium on Robotics Research (ISRR)*, 2013.

[10] A. Vezhnevets, J. Buhmann, and V. Ferrari, "Active learning for semantic segmentation with expected change," in *CVPR*, 2012.

[11] D. Wang, C. Yan, S. Shan, and X. Chen, "Active learning for interactive segmentation with expected confidence change," in *Asian Conf. on Computer Vision*, 2012.

[12] R. Triebel, H. Grimmett, and I. Posner, "Confidence boosting: Improving the introspectiveness of a boosted classifier for efficient learning," in *Autonomous Learning Workshop at ICRA*, 2013.

[13] B. Settles, *Active Learning*. Morgan & Claypool, 2012.

[14] P. Zhang, J. Wang, A. Farhadi, M. Hebert, and D. Parikh, "Predicting failures of vision systems," in *CVPR*, 2014.

[15] R. Paul, R. Triebel, D. Rus, and P. Newman, "Semantic categorization of outdoor scenes with uncertainty estimates using multi-class Gaussian process classification," in *IROS*, 2012.

[16] N. Lawrence, M. Seeger, and R. Herbrich, "Fast sparse gaussian process methods: The informative vector machine," *Adv. in neural inf. proc. systems*, vol. 15, pp. 609–616, 2002.

[17] K. Lai, L. Bo, X. Ren, and D. Fox, "A large-scale hierarchical multi-view rgb-d object dataset," in *ICRA*, 2011.

[18] J. Riihimäki, P. Jylänki, and A. Vehtari, "Nested expectation propagation for gaussian process classification with a multinomial probit likelihood," *Journal of Machine Learning Research*, vol. 14, 2013.

Stream-based Active Learning for Efficient and Adaptive Classification of 3D Objects

Alexander Narr[1] Rudolph Triebel[1,2] Daniel Cremers[1]

***Abstract*— We present a new Active Learning approach for classifying objects from streams of 3D point cloud data. The major problems here are the non-uniform occurence of class instances and the unbalanced numbers of samples per class. We show that standard online learning methods based on decision trees perform comparably bad for such data streams, which are however particularly relevant for mobile robots that need to learn semantics persistently. To address this, we use Mondrian forests (MF), a recent online learning algorithm that is independent on the data order. We present an extension of that algorithm and show that MF are less overconfident than standard Random Forests. In experiments on the KITTI benchmark, we show that this leads to a substantially improved classification performance for data streams, rendering our approach very attractive for lifelong robot learning applications.**

I. Introduction

One major requirement for most modern robotic systems is their ability to extract semantic information from their observed input data. In general, semantic information can be represented in many different ways, including drivable and non-drivable areas for outdoor robots [1], annotated road maps with lanes, intersections and road signs for self-driving cars [2], or the position and orientation of door knobs, hinges of doors for robots with manipulators [3]. In this work, we focus on class labels of road participants such as cars, pedestrians, or cyclists, although our approach can also be used for other applications. The main idea is that of *persistent* learning, i.e. the robot learns semantics with new observations during operation and not from a previously manually annotated data set. The benefits of this are two-fold: first, the robot is able to adapt to new situations as new observations directly modify its internal representations. And second, learning is done mainly on those observations that actually occur in the robot's environment and not on the much larger set of *potential* observations, i.e. it can be done more efficiently.

However, as we will show in this paper, these benefits come with a major difficulty: in contrast to an offline recorded training data set, data that is perceived online is highly correlated to previous observations. For example, an autonomous car driving on a highway may observe many other cars or motor cycles from many different view points, which can all be added to a large, growing training data set. This leads to a large, but very unbalanced data set, because other road participants such as cyclists or pedestrians

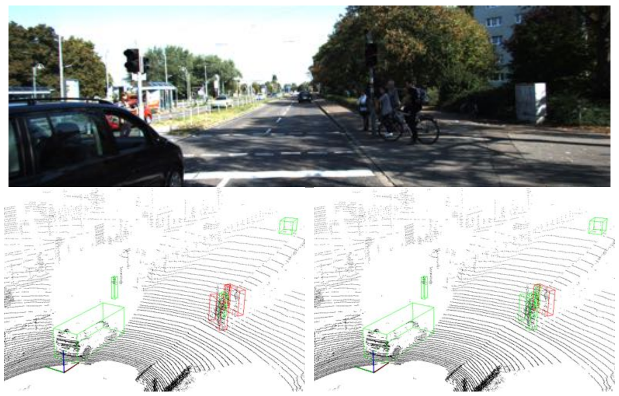


Fig. 1. Example situation from the KITTI benchmark data set with a car, a cyclist and some pedestrians. The 3D point cloud data of this frame was classified after actively learning semantics from a stream of 10 000 previous samples. The bottom left image shows the result from actively learning an online Random Forest classifier, the right one shows the result using a modified Mondrian forest instead. A green bounding box refers to a correct classification, while red boxes are wrong predictions. As we show in this paper, the better performance of Mondrian forests comes from their higher capability to learn from streams of data.

are under-represented. As a consequence, identifying new observations as instances of an unknown object class on appearance is very hard, because standard online learning methods are only able to specialize, but not generalize their models with new data samples and therefore require many samples from the new class. In the literature on Active Learning, this is known as the distinction between *pool-based* and *stream-based* learning. To address these problems, we propose an Active Learning framework that uses a more informed and more flexible classification algorithm based on a recent machine learning method. The so-called *Mondrian forest* has the major benefit that it is independent on the order in which the data appears, and that it can handle unbalanced data sets much better because its representation does not actually depend on the ground truth class labels. As we show experimentally, this leads to much steeper learning curves even for the stream-based learning scenario.

Our work consists of three major contributions: First, we provide modifications to the original Mondrian forest algorithm, which lead to a better classification performance. Second, we analyze Mondrian forests regarding their tendency to make wrong predictions with low uncertainty, i.e. their *overconfidence*, and we show that Mondrian forests tend to be less overconfident than standard online random forests. And third, we exploit this by setting up an *active learning* framework using Mondrian forests as underlying

[1]Computer Vision Group, Dep. of Computer Science, TU Munich, 85748 Garching, Germany `[narr,triebel,cremers]@in.tum.de`
[2]Institute of Robotics and Mechatronics, Dep. of Perception and Cognition, German Aerospace Center (DLR), 82234 Weßling, Germany `rudolph.triebel@dlr.de`

classifier. We show experimentally that this new Active Learning method produces less label queries at a higher prediction accuracy compared to online Random Forests, and that it is particularly well suited for stream-based learning problems such as those often encountered in robotics.

II. Related Work

Our work has links to three different learning concepts: incremental learning, online learning and Active Learning. Incremental learning refers to “any online learning process that learns the same model as [it] would be learnt by a batch learning algorithm” [4]. Essentially, this means that it can incorporate additional information from newly observed data, e.g. new object classes. Thus, the benefit over offline learning is that incremental learning requires less computation steps when new data is available. An example of an incremental learning method is Learn++ [5], an extension of AdaBoost. The idea is to generate new hypotheses from additional batches of training data and to update the ensemble of weak classifiers accordingly. Another incremental method are nearest class mean forests (NCMF) [6]. They consist of random forests where the decision nodes are based on nearest class mean classifiers [7]. In contrast to other random forests, NCMFs are able to add new classes, because they store parts of the underlying distribution of the feature space in each node. Another popular incremental learning approach is based on one-class Support Vector Machines (SVMs) [8], [9]. The key idea is that the current SVM model is updated on the new data and the support vectors from the previous learning step. The main drawback with all incremental learning methods is however, that they do not provide a functionality to update the learned model from a single new data observation, and they often require to store at least a large fraction of the training data.

In contrast, online learning methods only use the information from the next data sample, and they do not need to re-use any of the previously observed samples. Recent examples of online learning methods are given by the work of Saffari *et al.* [10], who introduced online multi-class Gradient Boost and online multi-class LP Boost. In contrast to offline boosting, an online boosting method has a fixed number of weak classifiers and the weak learners themselves are online algorithms, for example online Random Forests (ORF) [11]. The performance of these methods is very good, but as we will show in the experiments, they have problems when the training data is un-balanced.

Finally, our work makes use of an Active Learning framework. A good overview of this topic is given by Settles [12]. Some applications of Active Learning include the work of Kapoor *et al.* [13], who perform object categorization using a Gaussian Process classifier (GPC), the work of Vezhnevets *et al.* [14], as well as that of Wang *et al.* [15] who use active learning for interactive image segmentation. In contrast to all these approaches, we address the problem of stream-based Active Learning as opposed to pool-based learning. More details are given below.

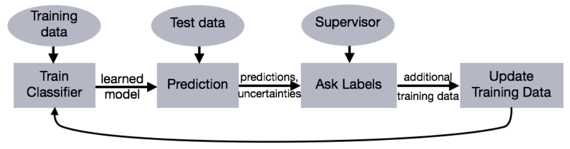


Fig. 2. Generic Active Learning. Starting with an initial training round, new test data are classified, resulting in label predictions and corresponding uncertainties. Based on these, a human supervisor is asked for ground truth labels, and these are joind with the current training data. Then, training is repeated with the extended training data until a stopping criterion is met.

III. Persistent Learning for Mobile Robots

Our goal is to develop an algorithm that learns semantic information (e.g. object class labels) from a large input data set in such a way that it *a)* adapts its internal representation to new, unseen environments, *b)* only requires few interactions with a human supervisor (“teacher”) and *c)* only costs little computational effort when considering a new data sample. The motivation for these aims arises from the intended application in mobile robotics: robots that persistently learn semantics must be able to cope with new situations, should only ask a human when necessary (thereby increasing their level of autonomy), and perform the required computations fast enough to not block the functionality of the whole system. To address the first two goals, researchers have investigated *active learning* methods, and we will briefly explain this in the following. The third goal refers to the capability for *online* computation, i.e. to update the representation without having to re-consider previously observed data samples. In the following, we give some examples for existing online learning methods. Finally, we highlight the major difference of active learning for mobile robots as opposed to other applications such as computer vision and explain the typical drawbacks of standard online learning methods for our purpose. This will motivate our proposed approach presented in Sec. IV.

A. Active Learning

Fig. 2 shows a schematic flow of a generic active learning framework. It starts with an initial training set $(\mathcal{X}_0, \mathcal{Y}_0)$, where $\mathcal{X}_0 = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ are N observations represented as d-dimensional vectors and $\mathcal{Y}_0 = \{y_1, \ldots, y_N\}$ are ground truth class labels, i.e. $y_i \in \{1, \ldots, C\}$ with $C \geq 2$. Then, in each learning round or *epoch* $j = 0, 1, \ldots,$ a classifier f_j is trained using $(\mathcal{X}_j, \mathcal{Y}_j)$. This gives a mapping $f_j : \mathbb{R}^d \to \mathbb{R}^C$ that assigns a prediction $\mathbf{p} \in \mathbb{R}^C$ to each input sample. Next, a set of new observations $\mathcal{X}_j^* = \{\mathbf{x}_1^*, \ldots, \mathbf{x}_K^*\}$ is considered and classified using f_j. The result are prediction vectors $\{\mathbf{p}_1^*, \ldots, \mathbf{p}_K^*\}$. The key step is then to select a sub set of $\mathcal{X}_j^*$ for a query of corresponding ground truth labels. A common selection method uses the prediction uncertainties to decide whether a query is triggered. From this query, new ground truth labels $\mathcal{Y}_j^*$ are obtained and, together with the selected observations, they are added to the current training data set $(\mathcal{X}_j, \mathcal{Y}_j)$. The resulting set $(\mathcal{X}_{j+1}, \mathcal{Y}_{j+1})$ is then used to train the classifier f_{j+1} in the next epoch.

B. Online Learning

In principle, active learning can be performed with any classification algorithm that is capable of providing uncertainty estimates with class predictions for new samples. However, in terms of efficiency standard offline learning is not good for Active Learning, because it requires retraining on all the data observed so far in every epoch. As a consequence, the time needed for learning steadily increases and all observed data samples must be kept in memory. Therefore, online algorithms for learning are much more useful. In the literature, many standard offline classification methods have been converted into online methods, but one remarkable contribution is the work of Saffari *et al.* [10], which provides very efficient online multi-class learning methods based on boosting. The key element in that work is an online Random Forest, which is used as a weak classifier in boosting. An online Random Forest combines online bagging, random feature selection and the growing strategy of extremely randomized trees [16], where the test functions and thresholds are generated randomly. In contrast to an offline node, an online decision node has to see a minimum number α of samples before splitting and a split has to achieve a minimum gain β. For data sets that are roughly uniformly sampled across the classes, this gives comparably good classification results, but for non-uniform data and for data streams this causes problems, as explained below.

C. Pool-based vs. Stream-based Learning

When we described the Active Learning framework, we deliberately did not specify the way in which the test data set $\mathcal{X}^*$ is given. In principle, there are two ways to do this: either $\mathcal{X}^*$ is a fixed-size set of observations that are collected beforehand, or it consists of a growing number of samples that are continously observed and added to $\mathcal{X}^*$. Thus, in the first case we have a fixed *pool* of data, and the algorithm can pick good samples to query from this pool, which is usually very large. This is the most common application for Active Learning. However, in mobile robotics, and in particular for robots that are to learn semantics persistently, the second scenario known as *stream-based* Active Learning is much more relevant, because robots perceive streams of data, and they should be able to learn from it continously. Therefore, in this paper we consider stream-based Active Learning.

To highlight this distinction further, we performed the following experiment. We considered a large, standard benchmark data set and applied two online classification methods: online Random Forests (ORF) and online multi-class Gradient Boost (OMCGB) with ORF as weak classifiers (for both methods see [10]). Then, we resampled the data in such a way that the occurence of samples in each class was distributed uniformly over the time line (see Fig. 3, left and center) and applied again the online classification methods. Evaluation was done on a hold-out set not used for training, and we choose the KITTI data [17] for this experiment (for more details on the used data set we refer to the description in Sec. V-A). The resulting learning curves are shown in the right plot of Fig. 3. As we can see, both

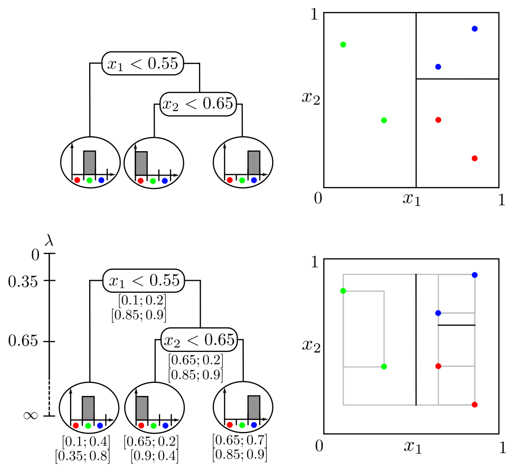


Fig. 4. Comparison between a decision tree and a Mondrian tree for a toy example with three different classes (Figure inspired by [18]). The feature space is defined in $[0, 1]^2$ where x_1 and x_2 denote horizontal and vertical axes. **Top:** The partition of the space of a decision tree with two axis aligned splits at $x_1 = 0.55$ and $x_2 = 0.65$. **Bottom:** Embedded partition of the space of a Mondrian tree where each node has an associated split time, and the splits are committed only within the range of each sub tree.

online learning methods perform resonably well for the case of uniform distributions of class occurences. However, on the original data, where many objects of the same class can appear for some time period but for others there are almost no occurences, we have a significantly worse performance of the online learners. The reason for this behavior is that online Random Forests can not handle well data sets with unbalanced classes and with a non-uniform distribution of class occurences. Therefore, in this paper we propose a different approach, which we describe next.

IV. Proposed Approach

The main drawback of online Random Forests is that their structure strongly depends on the order in which the data samples are observed. As soon as a split is made at a given leaf node – thereby creating two new leaf nodes, this split decision can not be revised. Also, ORFs can only grow at the leaves, i.e. new samples can only refine the model, but it can not be made more general by a new sample. A novel algorithm that explicitly addresses these problems is the *Mondrian Forest* by Lakshminarayanan *et al.* [18]. In this section, we briefly review this algorithm and present some modifications for an improved performance. Then, we analyse it with respect to its tendency to associate wrong classifications with a high uncertainty [19], a key feature for application in Active Learning.

A. Mondrian Forests

The major difference between a Mondrian tree and a standard decision tree is that Mondrian trees also store the *extent* of the data that corresponds to each node. A simple example with three classes is shown in Fig. 4. While the decision tree uses splits that range over the entire *potential* range of the data, the splits of a Mondrian tree only cover

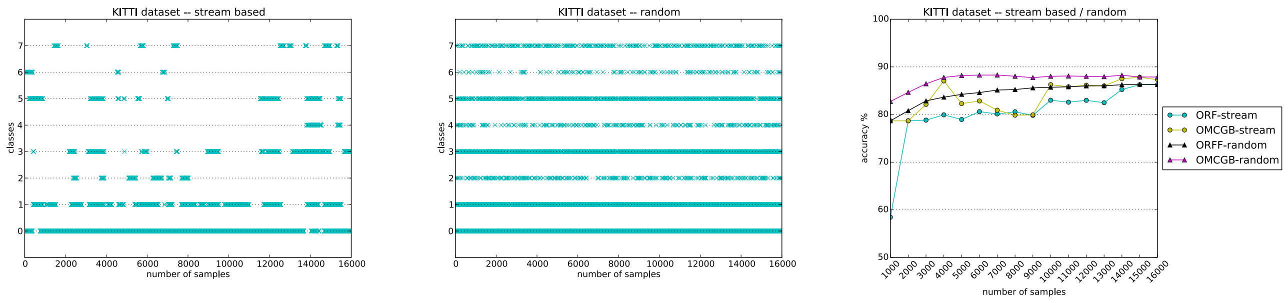


Fig. 3. **Left and center:** Visualization of class occurences in the KITTI data set [17]. The x-axis shows the time axis represented as number of observed samples, i.e. every new observation corresponds to a time step. On the y-axis we plot the ground truth class indices of the observed samples. Class 0 corresponds to 'car', 3 to 'pedestrian' and 5 to 'cyclist'. The left plot shows the situation of the original data, the center plot was created after uniformly re-sampling the data. **Right:** Learning curves of two standard online learning methods: online Random Forests (ORF) and online multi-class Gradient Boost (OMCGB), both evaluated on the original and the re-sampled data ("stream" vs. "random"). As we see, the performance of both methods for the original data stream is significantly worse than for the resampled set.

the *actual* data range. This is achieved by keeping bounding box information for each sub tree, i.e. each node v_j stores the lower and upper boundaries $\mathbf{l}_j$ and $\mathbf{u}_j$ of all data samples $\mathbf{X}_j = \{\mathbf{x}_{j,1}, \dots, \mathbf{x}_{j,n_j}\}$ associated with the sub tree at v_j, where n_j is the number of these samples. Note that the data itself is *not* stored in the tree, only the bounding boxes. As in random decision trees, splits are generated randomly, but samples are drawn from an exponential distribution whose rate parameter is proportional to the data extent of the sub tree. Concretely, each node v_j has an associated time step τ_j, and the entire tree has a time horizon or *budget* λ. Time steps τ_j increase with the depth of the tree and splits are only created as long as $\tau_j < \lambda$. Thus, λ implicitly regulates the depth of the tree. For clarity, Algorithm 1 shows in detail the recursive computation of a Mondrian tree from a given data set $\mathcal{D}_j = (\mathcal{X}_j, \mathcal{Y}_j)$, a node index j and a budget λ. Initially, j is set to the root node index and $\mathcal{D}_j$ contains the entire data set[1]. Then, in each call of `SampleMondrianBlock` a new node is created and the bounding box $(\mathbf{l}_j, \mathbf{u}_j)$ computed (in lines 3 and 4, upper indices i denote feature dimensions). If all labels of the sub tree are equal, no split is inserted, otherwise a new time parameter E is sampled and added to the time step $\tau_{\text{par}(j)}$ of the current parent node. Note that large values of e_j lead to a higher probability of small values for E, i.e. large bounding boxes are more likely to be split than smaller ones. If a split is inserted, the split dimension δ_j is sampled proportional to the extents of the dimensions, and the split location ξ_j is sampeled uniformly along this extent. Then, data is split into a left part $\mathcal{D}_{j_l}$ and a right part $\mathcal{D}_{j_r}$, and the corresponding sub trees are generated recursively.

B. Un-pausing Mondrian Blocks

In the original algorithm, splits are not inserted for nodes where all samples have the same label (see line 5 of Alg. 1). The authors call this "pausing" a Mondrian block, and they do this to make Mondrian forests "comparable" to standard Random Forests. However, from our experiments

Algorithm 1: SampleMondrianBlock$(j, \mathcal{D}_j, \lambda)$ [18]

```
1  $v_j \leftarrow$ CreateNewNode$(T, j)$
2  for $i = 1, \dots, d$ do
3      $\ell_j^i \leftarrow \min\{x_{j,1}^i, \dots, x_{j,n_j}^i\}$
4      $u_j^i \leftarrow \max\{x_{j,1}^i, \dots, x_{j,n_j}^i\}$
5  if AllIdentical$(\mathcal{Y}_j)$ then
6      $\tau_j \leftarrow \lambda$
7  else
8      $e_j \leftarrow \sum_i (u_j^i - \ell_j^i)$
9      $E \leftarrow$SampleExp$(e_j)$
10     $\tau_j \leftarrow \tau_{\text{par}(j)} + E$
11 if $\tau_j < \lambda$ then
12     $\delta_j \leftarrow$SampleProp$(u_j^1 - l_j^1, \dots, u_j^d - l_j^d)$
13     $\xi_j \leftarrow$ SampleUniform$(\ell_j^{\delta_j}, u_j^{\delta_j})$
14     $(j_l, j_r) \leftarrow$ MakeChildrenIndices$(j)$
15     $\mathcal{D}_{j_l} = \{(\mathbf{x}_{ji}, y_{ji}) : x_{ji}^{\delta_j} \leq \xi_j\}$
16     $\mathcal{D}_{j_r} = \{(\mathbf{x}_{ji}, y_{ji}) : x_{ji}^{\delta_j} > \xi_j\}$
17     SampleMondrianBlock$(j_l, D_{j_l}, \lambda)$
18     SampleMondrianBlock$(j_r, D_{j_r}, \lambda)$
19 else
20     $\tau_j \leftarrow \lambda$
21     AddToLeaves$(v_j, T)$
```

we found that this can cause a bad classification performance of the algorithm, especially if many samples from the same class are observed in a row. In that case, a large amount of these samples will be lost, because the trees only store the bounding boxes of the data. Therefore, if then samples from a new class arrive, the inserted split can not take these "lost" samples into account, which often leads to inadequate splits. To mitigate this problem we define a split threshold parameters θ_s, which determines the maximum number of samples with the same label that can be contained in a leaf node. Thus, we modify the conditon in line 5 into

$$\texttt{AllIdentical}(\mathcal{Y}_j) \wedge n_j < \theta_s. \tag{1}$$

[1]Note that Alg. 1 describes the offline method to build a Mondrian forest from a given data set. For brevity, we omit the description of the actual online method that updates the tree with a single new data sample [18].

C. Under- and Overconfidence

As mentioned above, our aim is to use Mondrian forests in an Active Learning framework to achieve better classification results for the important case of stream-based data. To do this, an important question is whether the classifier is able to provide useful confidence estimates with its label predictions. In essence, it should be avoided that the classifier makes wrong predictions with a high confidence (corresponding to a low uncertainty), because in that case thresholding on the prediction uncertainty will not lead to good label queries (see Sec. III-A). In Mund *et al.* [19] this was denoted as *overconfidence*, and a formal definition of over- and underconfidence was introduced, which we briefly revise here. Overconfidence is the average confidence of all incorrectly classified samples from a given test set $(\mathcal{X}^*, \mathcal{Y})^*$, and underconfidence is the average uncertainty of all correctly classified samples. Note that we define confidence as one minus the uncertainty of a prediction. Active Learning with a classifier that is overconfident leads to bad classification results, whereas underconfident classifiers reduce the learning efficiency by generating too many label queries. Note that this is independent on the actual classification performance of the classifier, which is often measured in precision and recall. Even a classifier that produces only very few wrong predictions can be overconfident if its uncertainty on these few incorrect predictions is too low. Such a classifier might be very useful for offline applications, but not as much for Active Learning as a non-overconfident but less accurate classifier might be. Therefore, to test for suitability of Active Learning, we need to analyse the classifier with respect to its over- and underconfidence, which we do in Sec. V-D.

V. Experimental Results

In this section, we show experimentally that Mondrian Forests perform better on stream-based data, that they tend to be less overconfident compared to standard Random Forests, and that our modified version of Mondrian Forests outperforms state-of-the-art multi-class online learning methods when used in an Active Learning framework[2]. Before, we describe the details about the data and the features we used.

A. Data Sets and Feature Computation

Experiments were carried out on two different data sets: one standard set that is frequently used in the machine learning community and one benchmark set from a real robotics application. The first one is named `pendigits` and consists of 5620 samples of handwritten digits where each sample is represented by 64 attributes and each of the ten digit classes contains roughly the same number of samples [20]. This comparably easy data set is very useful for comparison with other incremental and online learning methods. The second data set is from the KITTI benchmark and consists of 18 streams of segmented 3D point clouds from urban traffic environments [17], which we concatenate

[2]An implementation of our method is available under https://github.com/SpeedyN/StreamBasedAL.git

TABLE I
ARTIFICIAL TRAINING SETS WITH TWO NEW CLASSES IN EVERY ROUND

Classes	0	1	2	3	4	5	6	7	8	9
S_1	124	129	124	129	0	0	0	0	0	0
S_2	124	128	124	128	252	247	0	0	0	0
S_3	124	128	124	128	126	124	371	378	0	0
S_4	124	128	124	128	126	124	124	126	495	504
T	55	57	57	56	55	55	56	55	55	56

to one long stream. Each of the 25,090 segments corresponds to a 3D bounding box containing points that represent a given object candidate. For each such candidate, we compute a 60-dimensional feature vector as proposed by Himmelsbach *et al.* [21]. These features consist of global characteristics such as box volume and mean intensity, as well as of distributions of local features such as scatterness or flatness. Not only can these features be computed in real time, but they are also comparably low-dimensional (as opposed to HMP features [22], for example, with 14,000 or more dimensions). This is important when using Mondrian forests, because their additional memory requirements are particularly evident for high-dimensional features. For the test data set we split each of the 18 sub sets at the ratio 2/3 to 1/3 and obtain a stream of 16,000 training samples and a set of 9,090 test samples.

B. Adding new classes

In the first experiment, we artificially created a situation of newly observed classes. First, we divided the `pendigits` data set into five sub sets $S_1, \dots, S_4$ and T as shown in Table I (numbers correspond to occurences of samples per class). The sets $S_1, \dots, S_4$ consist of samples from a growing number of classes, while T was used for testing. Then, we applied a number of incremental and online learning methods, where each time we started training on S_1 and then increased the training set by the next sub set S_2, S_3, S_4 before re-training. This was done for two incremental learning methods, namely MSVDD [8] and MOCSVM [9], and four online learning algorithms, namely online Random Forests (ORF), online multi-class Gradient Boost (OMCGB), online multi-class LPBoost (OMCLP) [10]), and Mondrian forests (MF) [18]. The results are given in Table II. As we can

TABLE II
RESULTS FOR THE LEARNING SCENARIO OF TABLE I

Data sets	S_1	S_2	S_3	S_3
MSVDD	39.2 %	58.68 %	79.21 %	98.23 %
MOCSVM	39.8 %	59.25 %	77.95 %	95.18 %
Mondrian Forests	39.21 %	58.78 %	78.36 %	95.21 %
ORF	33.87 %	53.57 %	72.02 %	87.24 %
OMCGB	29.48 %	34.67 %	59.28 %	60.03 %
OMCLP	27.02 %	39.44 %	60.78 %	63.10 %

see, incremental learning methods generally perform better, which is no surprise as they can rely on more exploitation of the training data. However, from the online methods the Mondrian forests clearly perform best.

To increase the difficulty of the learning problem, we ran a second experiment, where only classes 0 and 1 were used for initial training. Then, in each learning round, we added

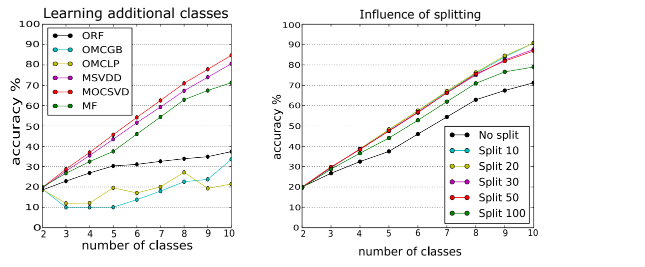


Fig. 5. **Left:** Results of incremental learning on "pendigits" . The experiment starts with samples from two classes and adds all samples from the next class in every round. **Right:** Experiments on the same data with varying thresholds θ_s to split nodes with uniform class labels ("un-pausing"). Performance is substantially increased using these artificial splits.

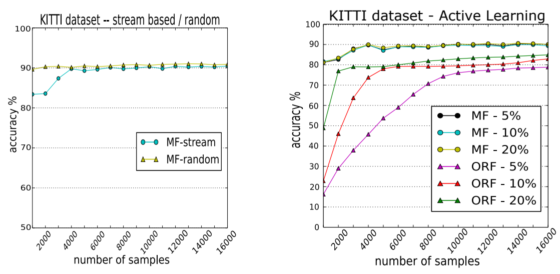


Fig. 7. **Left:** Learning curves of the Mondrian forest for the "re-sample" experiment from Sec. III-C. The MF classifier can deal much better with the data stream. **Right:** Classification accuracies for Active Learning using an MF and an ORF, where only 5%, 10% and 20% of the most uncertain data points are queried. Again, the MF clearly outperform the standard ORF.

all samples from the next class for re-training. This can be seen as the worst case scenario, and it is reflected by the bad performance of the standard online learning methods (see Fig. 5, left). Only the Mondrian forests can handle this hard case comparably well.

C. Benefit of Mondrian block unpausing

As mentioned in Sec. IV-B, we modify the original Mondrian forest algorithm in an important detail by artificially adding splits in nodes that consist only of samples with the same class label. To show the benefit of this, we plot the results for different values of the splitting threshold θ_s in Fig. 5 (right). We see that the earlier we decide to split these nodes, the better is the result. We note however that there is a trade-off with the depth of the trees, which leads to higher memory and run-time requirements. In practice, a value of $\theta_s = 20$ has proven to be a good compromise between efficiency and classification performance.

D. Under- and overconfidence

To evaluate under- or overconfidence of Mondrian forests we generated confidence histograms for correctly and incorrectly classified samples on a test set from the KITTI set (see Fig. 6). From the histograms in the first row, we see that in the beginning, when only few training samples are available (250 in our case), the ORF is much more overconfident than the MF, while underconfidence is not much different. In numbers we obtained overconfidences of 0.415 vs. 0.820 for the MF and the ORF respectively and 0.157 vs. 0.116 for the respective underconfidences. This difference is smaller later, when the training set consists of 16,000 samples (see bottom row in the figure). Here, the numbers are 0.529 and 0.583 for overconfidence and 0.126 and 0.132 for underconfidence of MF and ORF. However, for good classification results it is more important to reduce overconfidence already for small training sets, because then the samples that are incorrectly classified but detected as such have a stronger influence and can server better to improve the learned model.

E. Pool-based vs. stream-based Active Learning

In the last set of experiments, we evaluated the performance of the MF algorithm for the stream-based scenario. First, we ran the same experiment as described in Sec. III-C, and the result is given in Fig. 7 (left). We can see that the MF classifier increases its classification accuracy much faster than the ORF, even in the stream-based setting, and it also reaches a higher level (about 90% accuracy). We then tested the Active Learning scenario, where new label queries were generated after every 1,000 data samples. From these, we only used the most uncertain predictions for querying and re-training, and this fraction varied between 5% and 20%, i.e. from 50 to 200 samples per learning epoch. The result is shown in the right plot of Fig.7. The plot clearly shows that the MF can improve its classification accuracy even when trained only on a very small fraction of the data. Thus, on one side the MF classifier provides a higher level of learning autonomy by generating less queries and on the other side it can deal with the hard problem of learning from data streams. This is also reflected by the qualitative results in Fig. 8.

VI. Conclusions

In robotics, stream-based learning applications are much more relevant than standard pool-based approaches, because robots need to be adaptive to new environments. Data streams are however much harder to learn from, but we present a very effective and efficient approach to handle such situations. Based on a recent online learning algorithm, we show that Active Learning can be performed successfully on streams of data. While in our experiments we considered the problem to learn class labels, our approach is also useful for many other applications where semantic information must be automatically inferred from input data streams.

Acknowledgment: This work was funded by the EU project SPENCER (ICT-2011-600877).

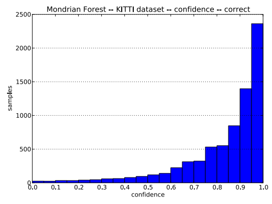
References

[1] P. Lamon, C. Stachniss, R. Triebel, P. Pfaff, C. Plagemann, G. Grisetti, S. Kolski, W. Burgard, and R. Siegwart, "Mapping with an autonomous car," in *IEEE/RSJ IROS Workshop: Safe Navigation in Open and Dynamic Environments*, 2006.

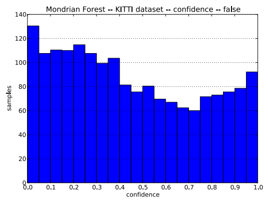
[2] H. Grimmett, R. Paul, R. Triebel, and I. Posner, "Knowing when we don't know: Introspective classification for mission-critical decision making," in *Int. Conf. on Rob. and Autom. (ICRA)*, 2013.

[3] T. Rühr, J. Sturm, D. Pangercic, M. Beetz, and D. Cremers, "A generalized framework for opening doors and drawers in kitchen environments," in *Int. Conf. on Rob. and Autom. (ICRA)*, 2012.

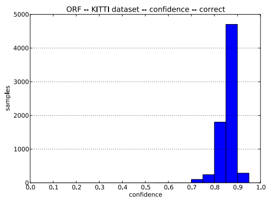
[4] C. Sammut and G. I. Webb, *Encyclopedia of Machine Learning*, 2010.



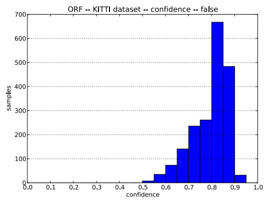
(a) MF - 250 - correct



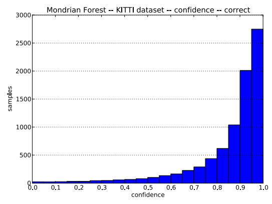
(b) MF - 250 - false



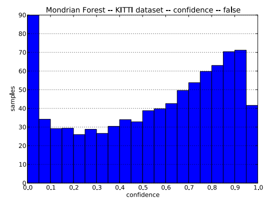
(c) ORF - 250 - correct



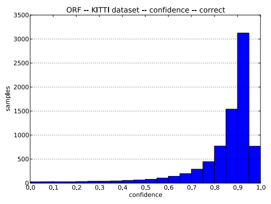
(d) ORF - 250 - false



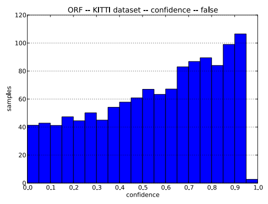
(e) MF - 16000 - correct



(f) MF - 16000 - false



(g) ORF - 16000 - correct



(h) ORF - 16000 - false

Fig. 6. Confidence values of a Mondrian forest (a, b, e, f) and an online Random Forest (c, d, g, h) on the KITTI data set. The upper row shows the confidence histograms after training on 250 samples and the lower row results after learning on 16,000 samples. We see that the MF is less overconfident, particularly in the beginning with little training data, as it makes wrong predictions with lower confidence (or, equivalently with higher uncertainty). At the same time, it is not more underconfident, as the correct predictions are mostly made with high confidence.

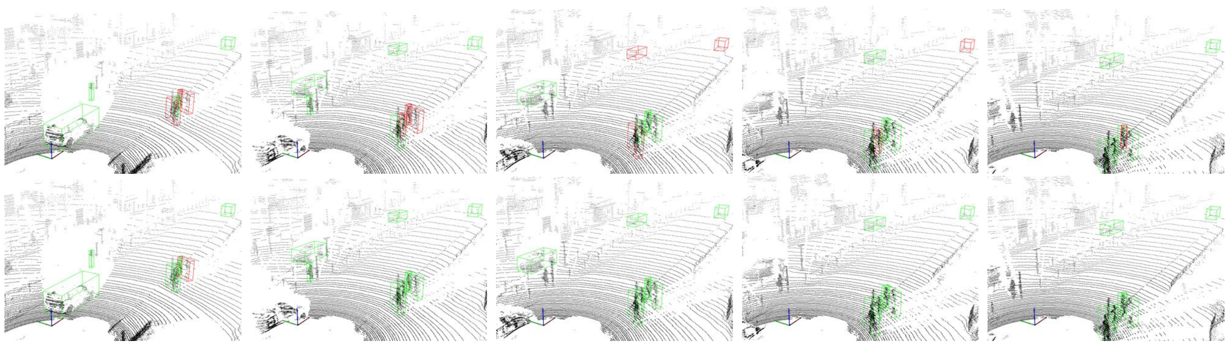


Fig. 8. Further results on the scene given in Fig. 1. From left to right, we show the next five data frames (point clouds), again after training on a stream of 10000 samples. The upper row shows the result for ORF, the lower row the MF result. Again, most classifications are done correctly by the MF (green boxes), while the ORF has many false classifications (red boxes).

[5] V. Polikar, Robi and Upda, Lalita and Upda, Satish S and Honavar, "Learn++: An incremental learning algorithm for supervised neural networks," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 31, pp. 497–508, 2001.

[6] M. Ristin, M. Guillaumin, J. Gall, and L. V. Gool, "Incremental Learning of NCM Forests for Large-Scale Image Classification," *Computer Vision and Pattern Recogn. (CVPR)*, pp. 3654–3661, 2014.

[7] T. Mensink, J. Verbeek, F. Perronnin, and G. Csurka, *Advanced Topics in Computer Vision*, ser. Adv. in Computer Vision and Pattern Recogn., G. M. Farinella, S. Battiato, and R. Cipolla, Eds. Springer, 2013.

[8] L. Yang, W.-m. Ma, and B. Tian, *Advances in Neural Networks*, ser. LNCS. Springer, 2011, vol. 6676, ch. New Multi-class Classification Method Based on the SVDD Model, pp. 103–112.

[9] A. K. N. Ho, N. Ragot, J. Y. Ramel, V. Eglin, and N. Sidere, "Document classification in a non-stationary environment: A one-class svm approach," in *Proc. of the Intern. Conf. on Document Analysis and Recognition (ICDAR)*, 2013, pp. 616–620.

[10] A. Saffari, M. Godec, T. Pock, C. Leistner, and H. Bischof, "Online multi-class lpboost," in *CVPR*, 2010.

[11] A. Saffari, C. Leistner, J. Santner, M. Godec, and H. Bischof, "On-line random forests," *Computer Vision Workshops (ICCV Workshops)*, pp. 1393–1400, 2009.

[12] B. Settles, *Active Learning*. Morgan & Claypool, 2012.

[13] A. Kapoor, K. Grauman, R. Urtasun, and T. Darrell, "Gaussian processes for object categorization," *Intern. Journal of Computer Vision*, vol. 88, no. 2, pp. 169–188, 2010.

[14] A. Vezhnevets, J. Buhmann, and V. Ferrari, "Active learning for semantic segmentation with expected change," in *CVPR*, 2012.

[15] D. Wang, C. Yan, S. Shan, and X. Chen, "Active learning for interactive segmentation with expected confidence change," in *Asian Conf. on Computer Vision*, 2012.

[16] L. Geurts, Pierre and Ernst, Damien and Wehenkel, "Extremely randomized trees," *Machine learning*, vol. 62, pp. 3–42, 2006.

[17] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the KITTI vision benchmark suite," in *Proc. of Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[18] B. Lakshminarayanan, D. M. Roy, and Y. W. Teh, "Mondrian Forests: Efficient Online Random Forests," in *Advances in Neural Information Processing Systems (NIPS)*, 2014.

[19] D. Mund, R. Triebel, and D. Cremers, "Active online confidence boosting for efficient object classification," in *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2015.

[20] E. Alpaydin and C. Kaynak, "Optical Recognition of Handwritten Digits Data Set," 1995.

[21] H. Himmelsbach, M. and Luettel, T. and Wuensche, "Real-time Object Classification in 3D Point Clouds Using Point Feature Histograms," in *Intern. Conf. on Intell. Robots and Systems (IROS)*, 2009.

[22] L. Bo, X. Ren, and D. Fox, "Hierarchical matching pursuit for image classification: Architecture and fast algorithms," in *Advances in neural information processing systems (NIPS)*, 2011, pp. 2115–2123.

Semi-supervised Online Learning for Efficient Classification of Objects in 3D Data Streams

Ye Tao Rudolph Triebel Daniel Cremers

Abstract— **We present a novel learning algorithm especially designed for challenging, large-scale classification problems in mobile robotics. Our method addresses two important aims: first it reduces the required amount of interaction with a human supervisor, which increases the level of autonomy of the learning process. And second, it has the capability to update its internal representation online with every new observed data sample, which makes it adaptive to new environments. The proposed method is based on a combination of two established methods, namely Online Star Clustering and Label Propagation, but it extends and modifies these in such a way that significant shortcomings such as classification inaccuracy and run time inefficiency can be resolved. In experiments on large benchmark data sets, we show that our approach can quickly learn to classify 3D objects with a significantly reduced amount of required ground truth labels for training.**

I. INTRODUCTION

With the advent of fast, high-resolution and at the same time affordable 3D sensors, mobile robotic platforms have recently experienced an enormous progress in terms of their perceptual capabilities. RGB-D cameras have become a standard equipment for mobile robots, and the recent achievements in RGB-D sensing suggest that these sensors will be the main exteroceptive sensor source in the near future. Their impact mainly stems from their high resolution, evidenced in very densely sampled resulting 3D point clouds, and from their ability to sense depth and color simultaneously and at high frame rates. However, while depth sensors are also used widely in other application areas such as surveillance tasks, in mobile robots there are at least two specific major challenges to solve. First, the large amounts of data produced by these sensors places particular problems for the learning algorithms used for automated semantic annotation tasks such as 3D object classification or semantic mapping. In principle, the major design goal for a mobile robotic platform is to be as autonomous as possible, i.e. interactions with human supervisors should be reduced to a minimal amount[1], and this includes interactions needed for learning, e.g. when labeling ground truth data for training. Therefore, to be autonomous, a robot should ask for semantic information only rarely, but this is hard when the acquired amount of data is large. And second, mobile robots usually operate in frequently changing environments, and they need to take decisions quickly and based on their current situation. Thus, the employed learning algorithms need to be *adaptive*, which means that many standard offline learning techniques are inadequate due to their computational requirements and their inability to modify internal representations on the fly.

Therefore, in this paper we propose a learning algorithm that simultaneously reduces the required amount of human effort in terms of providing ground truth label information, and operates online in the sense that it incorporates new information directly to update and refine its internal representation. The result is a fast and effective learning method that is particularly suited for semantic annotation of large 3D data streams, as we will show in the experiments. We achieve that using a novel online clustering algorithm that is particularly taylored for semi-supervised learning. Inspired by established graph-based methods such as Online Star Clustering [1] and online Affinity Propagation [2], our approach also uses an undirected, bipartite graph, however with the difference that non-exemplar nodes can not turn into exemplar nodes during vertex insertion, and that we use a more efficient nearest-neighbor search when inserting a new vertex. This and some other modifications make our clustering method more useful for subsequent label propagation, a fast and effective semi-supervised learning method. As we show in experiments on standard benchmark data, our method is able to efficiently learn 3D objects from large data streams online and with only little input from a human supervisor.

II. RELATED WORK

Our work combines semi-supervised learning (SSL) with online clustering and is therefore mostly related to these two areas[2]. For SSL, there is a good overview text book edited by Chapelle *et al.* [3], where detailed theoretical background is given, as well as a description of the most common techniques, including transductive Support Vector Machines (tSVM) [4], Gaussian Process classification (GPC) with null-category noise model [5] and Label Propagation [6]. Our proposed method mostly relates to the latter one, mainly due to efficiency reasons and because, as a graph-based approach, Label Propagation is very well suited for combination with efficient graph-based online clustering methods, which we employ in our approach. In that context, a number of earlier works have been proposed, where the most relevant ones are online k-means clustering [7], [8], online Expectation Maximization (EM) [9], [10], online affinity propagation (AP) [2], and online star clustering (OSC) [1]. The latter two approaches have the big advantage over k-means and EM

All authors are with Dep. of Computer Science, Technical University of Munich, Boltzmannstrasse 3 85748 Garching, Germany [ye.tao, rudolph.triebel, daniel.cremers]@in.tum.de

[1]Note that this is different from interactions with human *users*, where the robot itself provides a service and does not depend on human input.

[2]Note however the difference to *self-supervised* learning where semantic information comes from a different sensor source (see, e.g. [17])

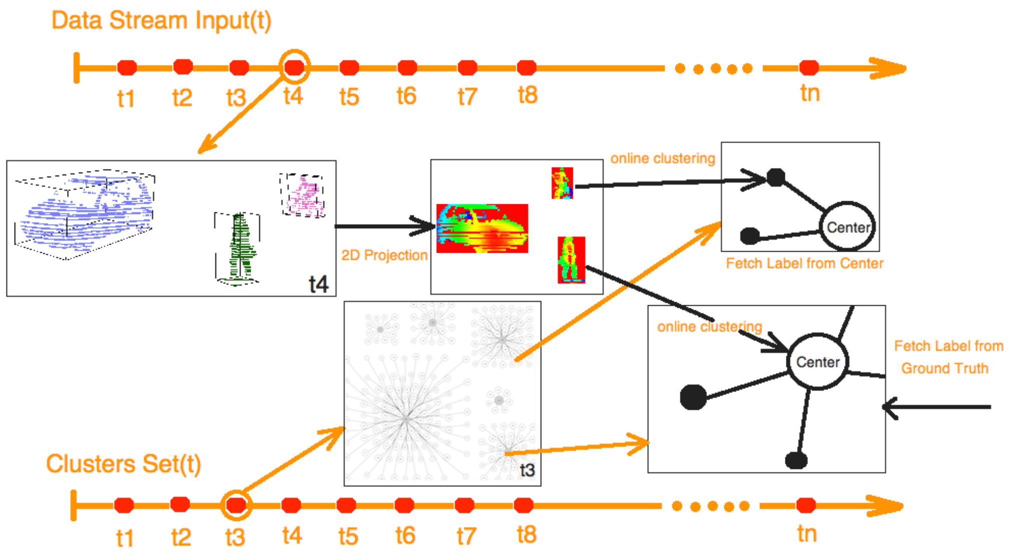


Fig. 1: Overview of our online learning approach for 3D object classification from data streams. See text for details.

that they do not require the number of clusters to be specified beforehand. Instead, they use a similarity threshold and determine the number of clusters implicitly. Our approach builds on OSC because it is more efficient than AP clustering (see also [11] for an application of OSC to unsupervised scene classification). However, as we will show, OSC has some drawbacks for our application in semi-supervised learning.

To compute features from 3D input data, we use Hierarchical Matching Pursuit (HMP) [12], an unsupervised learning algorithm based on sparse coding. In the area of unsupervised feature learning, many different approaches have been proposed, including Restricted Boltzmann Machines (RBM) [13], convolutional deep belief networks [14] and denoising autoencoders [15]. However, recent results on HMP-based classification [16], [12] show that they are very powerful and at the same time comparably efficient in learning. Therefore, we decided to use HMP features for our work.

III. Overview

Fig. 1 shows a schematic overview of our online 3D object classification method. We see two time lines: one on the top, which represents the incoming 3D point clouds at each time step, and one at the bottom for the status of the internal graph representation per time step. The depicted situation consists of an existing cluster graph at time step t_3 and a new sensor observation (point cloud) from the next time step t_4. In our pipeline, we first compute Hierarchical Matching Pursuite (HMP) features [16] for each pre-segmented point cloud in the current frame. We assume that the individual 3D objects are segmented with their 3D bounding box. Such a segmentation can be obtained from a tracking algorithm that separates moving objects from the static scene parts or by ground plane segmentation (see for example [18]). We note that, depending on the 3D sensor a pre-processing step might be required before feature computation. In the example here, data is obtained from a 3D laser scanner, which means that we have to create depth images from the point clouds before being able to apply HMP feature computation. Examples are shown in the center box in the upper part of the figure. Of course, when using RGB-D cameras, depth images are already available and need not to be computed explicitly.

The obtained HMP feature vectors are then inserted as vertices into the cluster graph. This graph distinguishes between center and satellite vertices, where the centers are *exemplars* for the satellites connected to them (details follow in Sec. V). Thus, a newly added vertex can either end up as a satellite of an already existing center, or it can be itself a new center. This is exemplified in the figure with the pedestrian and the cyclist, where the former builds a new center vertex, and the latter is associated to a new satellite. Then, the algorithm queries ground truth labels for the new centers if there are any, and infers class labels for the remaining vertices using Label Propagation. Our reasoning behind this is that centers are good potential representatives of an object class, particularly if they have many satellites attached, which by construction of the graph are similar to them. Thus, propagating labels from centers to satellites will lead to less misclassifications and fewer label queries than, e.g. propagating from satellites to centers. Note that the number of centers directly influences the performance of the algorithm: fewer centers lead to less label queries, making the learning algorithm more autonomous, but at a higher chance of misclassifications as more satellites will be different from their centers, i.e. the clusters will be less *pure*. The challenge is therefore to obtain pure, but few clusters (centers) at the same time. In Sec V we show how we address this trade-off, but first we consider a different approach combining two standard methods, and we show the drawbacks there that motivate our own method.

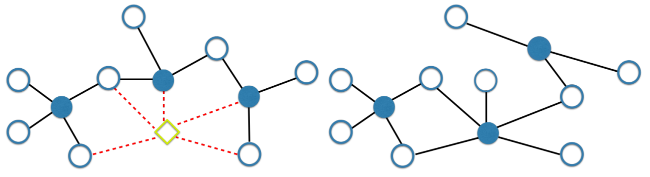

(a) vertex insertion (b) updated star graph

Fig. 2: Vertex insertion in the standard Online Star Clustering algorithm [1]. **(a)** A new vertex is inserted (yellow diamond) and all neighbors that are more similar than ϑ are determined (dashed lines). In this case, this results in a new center, because the degree of the new node is higher than those of the adjacent centers. **(b)** Rearranging requires removing and adding some edges and changing the role of some vertices.

IV. Online Semi-Supervised Learning

As a starting point, and also as a baseline for comparison with our proposed method, we consider here a straightforward combination of two concrete algorithms: Online clustering using the algorithm of Aslam *et al.* [1] and subsequent semi-supervised learning using Label Propagation [19].

A. *Online Star Clustering*

The main idea of the Online Star Clustering (OSC) algorithm [1] is to find a minimal number of maximal star-shaped subgraphs from a given thresholded similarity graph ("min-max criterion"). This means that the algorithm starts with a graph G_ϑ that consists of nodes v_i for each data sample and edges e_{ij} connecting two nodes that are more similar than a given threshold ϑ, i.e. $s(v_i, v_j) \geq \vartheta$ where s is a similarity measure. Then, it identifies some vertices as cluster *centers* and the remaining ones as *satellites* and removes all edges from G_ϑ that connect two satellites or two centers. The assignment of centers and satellites is made such that the number of clusters is minimal and the cluster sizes are maximal, and cluster centers have a higher degree (number of incident edges) than their connected satellites. For our application, the OSC algorithm has two major advantages over other clustering methods: First, it does not require the number of clusters to be given. Instead, its only parameter is the similarity threshold ϑ, which implicitly influences the number of resulting clusters. And second, the creation of the graph can be done online, i.e. after insertion of a single new vertex the min-max criterion is still valid. To guarantee this, in some cases the algorithm needs to re-assign centers and satellites and also to remove and add edges. An example of this is shown in Fig. 2.

The original OSC algorithm uses the cosine distance as a similarity measure between two connected vertices v_i and v_j, i.e.

$$s(v_i, v_j) = \frac{\langle v_i, v_j \rangle}{\|v_i\|\|v_j\|} = \cos(\alpha), \tag{1}$$

where α is the angle between the vectors that correspond to v_i and v_j. The authors show that with this similarity measure the similarity between two satellite vertices $u_i^{j_1}$ and $u_i^{j_2}$ that are connected to the same center c_i is bounded by

$$s(u_i^{j_1}, u_i^{j2}) = \cos\gamma \geq \cos\alpha_1 \cos\alpha_2 + \frac{\vartheta}{1+\vartheta} \sin\alpha_1 \sin\alpha_2, \tag{2}$$

where γ is the angle between the two satellites and α_1 and α_2 are the angles between the satellites and the center c_i. The interesting thing about this formulation is that it can be completely expressed in terms of dot products between feature vectors, provided that these are normalized. That means, we can also use other similarity measures instead by replacing dot products with *Mercer kernels*, for example the Gaussian kernel

$$k(v_i, v_j) = \exp\left(-\frac{\|v_i - v_j\|^2}{2\sigma^2}\right) \tag{3}$$

with a variance parameter σ, or the inverse city block kernel

$$cb(v_i, v_j) = \frac{1}{\sum_k |v_{ik} - v_{jk}| + \xi} \tag{4}$$

whith a constant parameter ξ. Thus, we then obtain a *kernelized* OSC algorithm. In the experiments (Sec. VI), we show that an appropriate kernel can substantially improve the measure of similarity.

B. *Label Propagation*

The OSC algorithm is a classical unsupervised learning method, i.e. it does not incorporate ground truth information for learning. However, in our application, we aim for automated semantic annotation, and this information can only come from some human supervisor. Therefore, we combine clustering with a semi-supervised learning (SSL) method by assigning ground truth labels to the center vertices and infering the labels for the unlabeled satellite vertices. In particular, we use Label Propagation (see [19], Algorithm 11.2). This method first computes an affinity matrix W where the entries are the node similarities, i.e. $w_{ij} = s(v_i, v_j)$ and it sets $w_{ii} = 0$. It then chooses a parameter $\alpha \in (0, 1)$ and a small $\epsilon > 0$ and computes $\mu := \alpha/(1 - \alpha)$. With this, it iterates over all vertices and updates the labels $\mathbf{y}_i$ of the vertices v_i in every iteration. We model class labels as vectors of a fixed length K, which determines the number of classes. A vertex v_i has then the class label k if $y_i^k = 1$ and all other entries of the vector $\mathbf{y}_i$ are zero. For unlabeled vertices, $\mathbf{y}_i$ is equal to the zero vector. The operations performed in each iteration of Label Propagation are as follows: If v_i is a labeled (center) node with an associated ground truth label $\mathbf{y}_i^{(0)}$ then the update rule at iteration $t = 1, 2, \ldots$ is

$$\mathbf{y}_i^{(t+1)} \leftarrow \frac{\sum_j W_{ij}\mathbf{y}_j^{(t)} + \frac{1}{\mu}\mathbf{y}_i^{(0)}}{\sum_j W_{ij} + \frac{1}{\mu} + \epsilon}. \tag{5}$$

If v_i is an unlabeled satellite vertex, then the rule is

$$\mathbf{y}_i^{(t+1)} \leftarrow \frac{\sum_j W_{ij}\mathbf{y}_j^{(t)}}{\sum_j W_{ij} + \epsilon}. \tag{6}$$

These rules are computed until a convergence criterion is reached. The described Label Propagation (LP) algorithm is formulated as an offline algorithm, although one could think of an extension to the online case. However, due to the shortcomings of this combined 'OSC+LP' approach, which we describe next, we do not consider an online LP version, but instead suggest an improved algorithm in Sec. V.

C. Drawbacks of the OSC+LP Approach

The presented combination of OSC and LP has at least three major drawbacks: First, it requires a nearest-neighbor search for each newly inserted vertex over the entire existing data set. This increases the run time significantly when the data set is very large. Second, due to the requirement that the min-max-criterion has to be fulfilled always, it can lead to many changes from satellite vertices to center vertices and vice-versa. This is not only inefficient in terms of computation time, but it also causes more label queries, especially if satellites turn into centers. In a sense, the fact that OSC guarantees a minimal number of clusters remedies this somehow, because with fewer clusters there are less label queries. However, as we perform label queries in every time step, one has to consider all vertices that at some point in the past have been centers, i.e. this includes all those satellites that where flipped from centers. And the third problem with OSC+LP is that the algorithm uses a fixed similarity threshold ϑ, which results in many isolated vertices that are not connected to any other one. These outliers again increase the required number of label queries, because each of them is considered a center of a cluster with size 1. In the next section, we propose our improved version of the algorithm, which particularly mitigates these three drawbacks.

V. Proposed Approach

Assume we are given a stream of data points $\mathbf{x}_1, \mathbf{x}_2, \ldots$ with $\mathbf{x}_i \in \mathbb{R}^d$. Our aim is to incrementally build from these data a graph $G = (\mathcal{V}, \mathcal{E}, \mathcal{Y}, P, S)$, where $\mathcal{V}$ is the set of vertices, $\mathcal{E}$ are the edges, $\mathcal{Y}$ are the class labels for each vertex, P are the *properties* of the vertices and S are the *similarities* assigned to each edge. Concretely, each vertex $v_i \in \mathcal{V}$ can have either the property 'center' or 'satellite', and each edge $e_{ij} \in \mathcal{E}$ connecting vertices v_i and v_j has a similarity value s_{ij} based on a distance measure between v_i and v_j. The center vertices play the role of *exemplars*, i.e. they are representatives of a whole set of other vertices, namely the satellites connected to them. In any stage of the algorithm, the graph G is bipartite, i.e. there are only edges connecting centers with satellites, and never edges between two centers nor between two satellites. The main idea of our semi-supervised online learning algorithm is to use only the centers to query ground truth class labels and to use these to infer the labels for the satellites, i.e. in a similar way as is done in label propagation. The individual steps of our algorithm are described next.

A. Vertex Insertion

As mentioned, our algorithm is inspired by the Online Star Clustering (OSC) approach [1]. This means, we also aim for a vertex insertion method that is efficient and at the same time produces graphs that correspond to a good data clustering. However, in standard OSC, the major focus is laid more on the latter by guarateeing that after insertion the graph still consists of a minimal set of maximal star-shaped sub-graphs (i.e. clusters). While this is a good property for pure unsupervised learning applications, in semi-supervised learning as we propose it, it is more advantageous to have *purer* clusters, even if the number of clusters is not minimal. Concretely, this means that we do not stricly connect a new vertex to its closest neighbors in the graph as in OSC, thereby accepting that the number of clusters can be sub-optimal. This slight drawback is however outweighed by the fact that our insertion is more efficient and produces purer clusters.

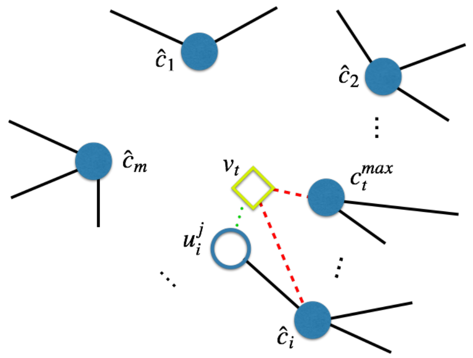


Fig. 3: Vertex insertion into the graph G. At time t, we insert vertex v_t (yellow diamond). For that, we first compute the m center vertices $\hat{c}_1, \ldots, \hat{c}_m$ that are closest to v_t (blue filled circles). Each of these centers has satellite vertices attached, here indicated with lines. The center closest (or most similar) to v_t is denoted c_t^{max}. Then, for $i = 1, \ldots, m$ we compute the similarities $s(v_t, \hat{c}_i)$ and $s(v_t, c_t^{max})$ (dashed red lines), as well as the similarities between v_t and all satellites u_i^j attached to center $\hat{c}_i$ (dotted green line). All such satellites that fulfill the condition in Eq. (8) are detached from $\hat{c}_i$ and connected to v_t, which then becomes a center vertex.

For the description of our insertion method, we define the set of centers $\mathcal{C}_t$ at time t as $\mathcal{C}_t = \{v_i \in \mathcal{V}_t \mid p_i = \text{'}center\text{'}\}$ and its cardinality as C_t. With this, the first step of insertion is to find a fraction $q \in [0, 1]$ of centers in $\mathcal{C}_{t-1}$ that are most similar to the new vertex v_t. This has two advantages: First, it does not require a fixed similarity threshold for connecting vertices, which avoids un-connected outliers. And second, it is more efficient than finding neighbors in the entire set of vertices $\mathcal{V}_t$, because $\mathcal{C}_t$ is usually much smaller than $\mathcal{V}_t$. Thus, formally we find an index ordering $\pi : \mathbb{N} \to \mathbb{N}$ so that $s(c_{\pi(i)}, v_t) > s(c_{\pi(i+1)}, v_t)$ for all $i = 1, \ldots, C_t - 1$, where we denote the similarity s as a binary function of vertices, and the elements of $\mathcal{C}_t$ are $c_1, \ldots, c_{C_t}$. Furthermore, we define the center that is most similar to v_t as c_t^{max}, i.e. $c_t^{max} := c_{\pi(1)}$. Then, the result of this first step is a subset $\hat{\mathcal{C}}_t \subset \mathcal{C}_t$ consisting of the first m sorted centers, i.e. $c_{\pi(1)}, \ldots, c_{\pi(m)}$ where $m = \lfloor qC_t \rfloor$.

In the next step, we search for neighbors of v_t in the set of satellites that are connected to centers in $\hat{\mathcal{C}}_t$. Thus, for each of the m most similar centers $\hat{c}_1, \ldots, \hat{c}_m$ we loop over all attached satellites u_i^j and decide whether they should be detached from their center $\hat{c}_i$ and reconnected to v_t, which then becomes a new center, or whether v_t is simply added as a satellite to c_t^{max}. Our criterion for this re-connection step is based on the similarity of the satellites u_i^j and v_t, but also on the general similarity of the cluster represented by $\hat{c}_j$. Concretely, we compute the *normalized similarity*

$$\bar{s}(v_t, \hat{c}_i) := \frac{s(v_t, \hat{c}_i)}{s(v_t, c_t^{max})} \tag{7}$$

and do a re-connection whenever

$$s(v_t, u_i^j)\bar{s}(v_t, \hat{c}_i) > s(u_i^j, \hat{c}_i). \tag{8}$$

If the condition in Eq.(8) is not valid for any satellite u_i^j, we connect v_t as a new satellite to c_t^{max}.

B. Minimal Average Cluster Size

As mentioned in Sec. IV-C, one major problem with standard OSC is that it tends to produce many unconnected vertices, which are treated as centers. This means that the SSL algorithm will query labels for them, even though the obtained information is often not very useful, because the queried labels are not representative for many other samples. The same problem occurs in our approach when many satellites of a given cluster center are detached and reconnected as described. Therefore, we introduce a parameter ω_{min}, which defines a lower bound on the average cluster size. Then, each time a satellite u_i^j should be detached from its center $\hat{c}_i$ according to Eq. (8), we test whether the average cluster size $\bar{z}_t$ is still larger than w_{min}. If this is not the case, all vertices of the cluster represented by $\hat{c}_i$ are inserted into the cluster that is closest to $\hat{c}_i$. Note that $\bar{z}_t$ at time t can be easily computed from $\bar{z} = \frac{t}{C_t}$, because at time t there are t vertices inserted in total and the number of clusters is C_t. The same procedure is done with the potentially new cluster formed by v_t as a center: As long as adding this new cluster does not cause the average cluster size $\bar{z}$ to be smaller than ω_{min}, it can be added. Otherwise, it is not added and v_t is attached to its closest center, as above. To avoid unnecessary re-connecting steps, we therefore wait until all centers closest to v_t are processed before we actually perform the reconnection of satellites.

C. Label Propagation

As in the OSC+LP approach described above, our last step is also label propagation. That is, if the newly added vertex v_t ends up as a new center, we query a ground truth label for it and propagate this new label to the satellites attached to v_t. If v_t has become a satellite, we propagate the label of the corresponding center (which was queried earlier queried) to it. Here, we note an important difference to the OSC algorithm: in our approach, all satellite vertices are connected to exactly one center. Therefore, Eq. (6) simplifies to a simple "copy" of the label from the center to the satellites. Similarly, Eq. (5) directly assigns the ground truth label to the labeled center. Thus, by our graph construction, the LP method can also be performed more efficiently.

D. Time-dependant Paramters

Two main parameters of our algorithm are the fraction q of most similar centers considered and the lower bound ω_{min} on the average cluster size. For both, there is an implicit dependence on the current size of the graph. In the beginning, there are only few samples and the graph is sparse. Thus, the fraction q of nearest neighbors can be larger, because nearest-neighbor search will anyhow be very efficient. Similarly, the minimal number of elements ω_{min} of an average cluster can be larger when there are more vertices in the graph. Therefore, we recompute q and ω_{min} at time step t as

$$\omega_{min}^t = \omega_{min}(1 - e^{-\tau * t}) \tag{9}$$
$$q^t = q(1 - e^{-\tau * t}), \tag{10}$$

where τ is a damping parameter.

Algorithm 1 summarizes all steps of our approach[3]. Note that satellites are not actually disconnected until all centers in $\hat{\mathcal{C}}$ have been considered and the new vertex forms a cluster that is large enough (line 15-17). Clusters that are too small are removed and all elements are assigned to the cluster that is most similar to their center. This is the $Recluster$ step.

Algorithm 1: Online Exemplar Based Clustering

```
   Data: stream $\mathbf{x}_t$ for $t = 1, 2, \ldots$
   Input: nearest-neighbor fraction $q$, damping factor $\tau$,
          min average cluster size $\omega_{min}$
   Output: inferred or queried labels $y_t$
1  $v^t \leftarrow CreateVertex(\mathbf{x}^t)$
2  $\omega_{min}^t \leftarrow \omega_{min}(1 - e^{-\tau t})$ (see Eq.(9))
3  $q^t \leftarrow q(1 - e^{-\tau t})$ (see Eq.(10))
4  $\hat{\mathcal{C}}_t \leftarrow MostSimilarExemplars(\mathcal{C}^t, v^t, q)$
5  $\bar{z} \leftarrow \frac{t}{C_t}$
6  $\mathcal{K} \leftarrow \emptyset$
7  forall the $\hat{c}_i \in \hat{\mathcal{C}}_t$ do
8      forall the $u_i^j$ connected to $\hat{c}_i$ do
9          if Eq. (8) is true then
10             $\mathcal{K} \leftarrow \mathcal{K} \cup u_i^j$
11             $MarkDisconnected(u_i^j, c_i)$
12     if $\bar{z}_t > \omega_{min}^t$ & $Size(\hat{c}_i) < \omega_{min}^t$ then
13         $Recluster(\hat{c}_i, \{u_i^j\})$
14 if $\bar{z}_t > \omega_{min}^t$ & $|\mathcal{K}| + 1 > \omega_{min}^t$ then
15     $DisconnectAllMarked(\{u_i^j\}, \{\hat{c}_i\})$
16     $MakeNewCluster(v_t, \mathcal{K})$
17     $y_t \leftarrow QueryGroundTruthLabel(v_t)$
18     $y_t \leftarrow PropagateLabels(v_t, \mathcal{K})$
19 else
20     $UnmarkAll(\{u_i^j\}, \{\hat{c}_i\})$
21     $Connect(v_t, c_t^{max})$
22     $y_t \leftarrow PropagateLabel(c_t^{max}, v_t)$
```

[3]An implementation of our approach in C++ is available at `https://github.com/mmmonkeytao/oscl.git`.

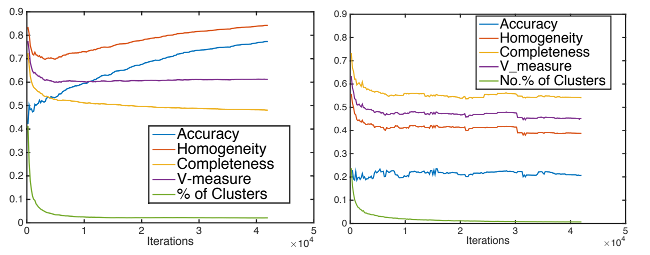


Fig. 5: Evaluation on the RGB-D data set based on V-measure (violet), classification accuracy (blue) and number of clusters per vertex (green). **Left:** Our approach. **Right:** OSC+LP. Note that our approach significantly outperforms the OSC+LP method in terms of accuracy and V-measure, although there is no big difference in the number of clusters.

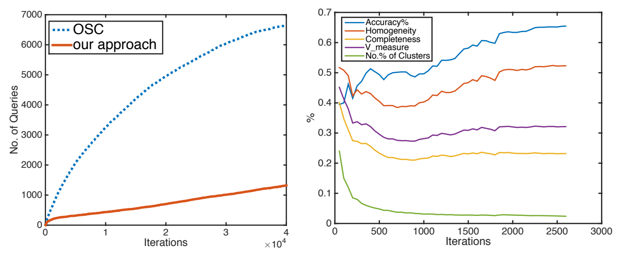


Fig. 6: **Left:** Accumulated number of generated label queries. Our approach generates significantly less label queries. **Right:** Result of our algorithm on the KITTI data set. The accuracy is worse than on the RGB-D set, but the input features are only based on depth values and not on color.

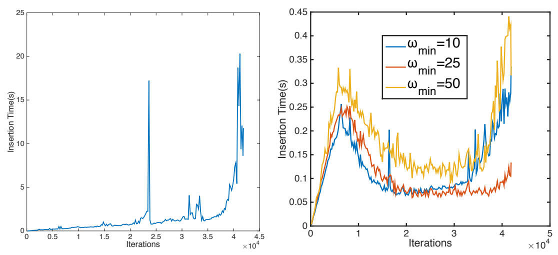


Fig. 7: Vertex insertion times for OSC (left) and our online clustering approach (right). Note that our approach never takes more than 0.5 seconds and that run time increases initially, because more new clusters are created. Later, new vertices often are satellites, which reduces the run time.

VI. Experiments

We evaluated our approach on two different data sets: the RGB-D data set provided by Lai *et al.* [20] with 41876 instances of 51 different object classes, and a subset of the KITTI data set of 3D point clouds in an urban environment [21], which contains objects from 7 classes, namely 1265 cars, 775 cyclists, 1035 Pedestrians, 957 vans, 667 trucks, 223 sitters and 257 bakground objects (misc). For both data sets, we compute HMP features, while for point clouds we first compute depth images (see Sec. III). For the HMP features, we first learn a dictionary of size 75 on the first level with K-SVD for depth and gray channels, and of size 150 for normal vectors and RGB channels. Then, on the second level we learn dictionaries of size 500 for gray scale and depth, and 1000 for RGB and normals. For each RGB-D image we compute an HMP feature vector of length 42000.

A. Similarity Analysis

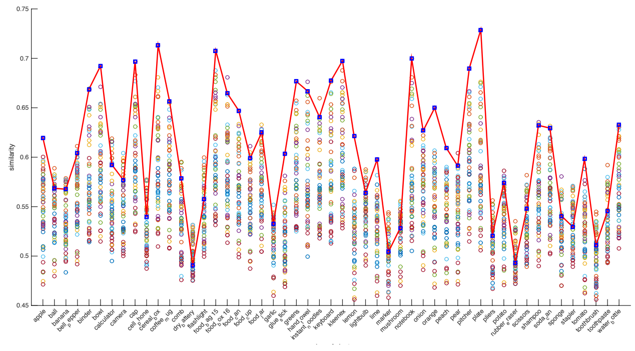
To find a good similarity measure (kernel) for our online clustering algorithm, we ran a specific test on 10,000 samples from the RGB-D data set. For each pair of images within the same object class and across different object classes we computed similarities and the corresponding average similarities. The result for the Gaussian kernel and the Inverse City Block (ICB) kernel are shown in Fig. 4. For each class, a colored circle refers to the average similarity with another class. Blue circles, which are connected with red lines depict the average self-similarity of each class. Thus, we can see that the self-similarity values tend to be better for the ICB kernel than for the Gaussian kernel. Therefore, in our following experiments, we only used the ICB kernel.

B. Online Learning of 3D Objects

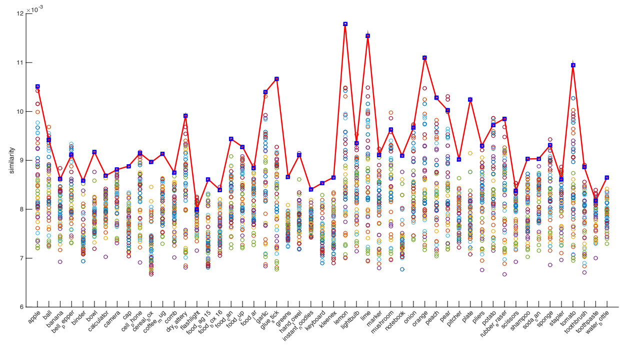
To assess the performance of our approach we randomly shuffle the 41876 different cropped images from the RGB-D data set and present them to our online SSL algorithm. We compare the results with the baseline method OSC+LP, described in Sec. IV, where we use the following criteria. First, the V_1-measure [22], which is a measure for cluster quality and consists of the harmonic mean between *homogeneity* and *completeness*. Intuitively, homogeneity is closely related to purity, i.e. it is high if clusters consist of many samples from the same object class. In contrast, completeness is high if many samples from an object class are in the same cluster (for details see [22]). Our second criterion is the number of clusters divided by the total number of vertices. Ideally, this value should be low because then we have less label queries. Finally, the third criterion is the classification accuracy, i.e. the percentage of correctly classified samples.

Fig. 5 shows the results on the whole RGB-D data set, where the left part shows our results, with $\omega_{min} = 25$, and the right part the ones obtained with OSC+LP with a threshold $\vartheta = 0.009$. As we can see, our method outperforms the OSC+LP algorithm both in terms of cluster homogeneity and in final classification rate. At the same time, the number of clusters produced by our algorithm is only slighty higher. This is good, because the number of clusters is directly related to the number of label queries. This is shown in Fig. 6 (left), where we plot the accumulated number of generated label queries for both methods. We see that, compared to OSC+LP our approach only requires very few ground truth labels for learning. For the KITTI data we obtain the results of Fig. 6 (right). Note that we only use a subset of the data because the entire data set is very unbalanced between the classes. We see that the classification is worse than the one on RGB-D, but the feature vectors only contain depth values.

Furthermore, we compare our approach with OSC+LP in terms of run time needed for a vertex insertion, as this was also one of our main design goals. Fig. 7 (left) shows the



(a) Gaussian Kernel with $\sigma = 1.0$



(b) Inverse CityBlock Kernel

Fig. 4: Comparison of two similarity measures (kernels). For each object class in the RGB-D data we show the average similarity across classes as colored circles, and the self-similarity for each class (blue circles, connected with red lines).

insertion time for OSC+LP for each iteration (insertion) on the RGB-D data. We can see one major peak at around 23000 observed samples. We relate this to an extremely large amount of cluster rearrangements required at this stage. Later, towards the end of the data set, the run time increases again very quickly. In contrast, insertion in our algorithm never takes more than 0.5 seconds (see right plot in Fig. 7).

VII. Conclusions

As we have shown, the combination of semi-supervised learning methods with online clustering can be a very efficient approach for learning 3D object classes from large data streams. However, a straightforward implementation using standard Online Star Clustering and Label Propagation results in a suboptimal performance, both in run time and in accuracy, because OSC is not particularly designed for combination with SSL. In contrast, if we modify the clustering algorithm accordingly, we obtain impressive results for 3D object classification with comparably little effort in terms of run time and generated label queries. As a consequence, our proposed method is very well suited for challenging online classification tasks in mobile robotics.

Acknowledgement: The work in this paper was funded by the EU project SPENCER (ICT-2011-600877).

References

[1] J. Aslam, E. Pelekhov, and D. Rus, “The star clustering algorithm for static and dynamic information organization,” *Journal of Graph Algorithms and Applications*, vol. 8, no. 1, pp. 95–129, 2004.

[2] J.-P. Zhang, F.-C. Chen, L.-X. Liu, and S.-M. Li, “Online stream clustering using density and affinity propagation algorithm,” in *Software Engineering and Service Science (ICSESS)*, 2013, pp. 828–832.

[3] O. Chapelle, B. Schölkopf, and A. Zien, Eds., *Semi-Supervised Learning*. MIT Press, 2006.

[4] T. Joachims, “Transductive inference for text classification using support vector machines,” in *Proc. of the International Conference on Machine Learning (ICML)*, 1999, pp. 200–209.

[5] N. D. Lawrence, J. C. Platt, and M. I. Jordan, “Extensions of the informative vector machine,” in *Proc. of the First Intern. Conf. on Deterministic and Statistical Methods in Machine Learning*. Springer-Verlag, 2004, pp. 56–87.

[6] X. Zhu and Z. Ghahramani, “Learning from labeled and unlabeled data with label propagation,” Carnegie Mellon University, Tech. Rep. CMU-CALD-02-107, 2002.

[7] W. Barbakh and C. Fyfe, “Online clustering algorithms,” *International Journal of Neural Systems (IJNS)*, vol. 18, no. 3, pp. 1–10, 2008.

[8] A. Choromanska and C. Monteleoni, “Online clustering with experts,” in *ICML W. on Online Trading of Exploration and Exploitation*, 2011.

[9] O. Cappé, “Online EM algorithm for hidden markov models,” *J. of Comput. and Graphical Statistics*, vol. 20, no. 3, pp. 728–749, 2011.

[10] S. Yildirim, S. S. Singh, and A. Doucet, “An online expectation–maximization algorithm for changepoint models,” *J. of Comput. and Graphical Statistics*, vol. 22, no. 4, pp. 906–926, 2013.

[11] R. Triebel, R. Paul, D. Rus, and P. Newman, “Parsing outdoor scenes from streamed 3d laser data using online clustering and incremental belief updates,” in *Robotics Track of AAAI Conference on Artificial Intelligence*, 2012.

[12] L. Bo, X. Ren, and D. Fox, “Hierarchical Matching Pursuit for Image Classification: Architecture and Fast Algorithms,” in *Advances in Neural Information Processing Systems (NIPS)*, December 2011.

[13] R. Salakhutdinov and G. Hinton, “Deep Boltzmann machines,” in *Proc. of the International Conference on Artificial Intelligence and Statistics*, vol. 5, 2009, pp. 448–455.

[14] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng, “Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations,” in *Proc. of the 26th Annual International Conference on Machine Learning*. ACM, 2009, pp. 609–616.

[15] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, “Extracting and composing robust features with denoising autoencoders,” in *Proc. of the 25th International Conference on Machine Learning*. New York, NY, USA: ACM, 2008, pp. 1096–1103.

[16] L. Bo, X. Ren, and D. Fox, “Unsupervised Feature Learning for RGB-D Based Object Recognition,” in *ISER*, June 2012.

[17] J. Maye, R. Triebel, L. Spinello, and R. Siegwart, “Bayesian online learning of driving behaviors,” in *Int. Conf. on Robotics and Automation (ICRA)*, 2011.

[18] O. H. Jafari, D. Mitzel, and B. Leibe, “Real-Time RGB-D based People Detection and Tracking for Mobile Robots and Head-Worn Cameras,” in *Int. Conf. on Robotics and Automation (ICRA)*, 2014.

[19] Y. Bengio, O. Delalleau, and N. L. Roux, *Semi-supervised Learning*. MIT Press, 2006, ch. Label Propagation and Quadratix Criterion.

[20] K. Lai, L. Bo, X. Ren, and D. Fox, “A large-scale hierarchical multi-view rgb-d object dataset,” in *Int. Conf. on Robotics and Automation (ICRA)*, 2011.

[21] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, “Vision meets robotics: The kitti dataset,” *Intern. Journal of Robotics Research (IJRR)*, 2013.

[22] A. Rosenberg and J. Hirschberg, “V-measure: A conditional entropy-based external cluster evaluation measure,” in *Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, 2007, pp. 410–420.