Profile Project AI
Transfer learning using an automatically generated shared feature space

Authors:

Agnes van Belle *(10363130)*,
Jorge Sáez Gómez *(10524924)*,
Lydia Mennes *(10333843)*

July 7, 2013

Abstract

Transfer learning is a machine learning framework that considers itself with the case where we would want to re-use knowledge from a learning task in another setting where the task or domain might have a different feature space and/or distribution. In the case of domain adaptation or transductive transfer learning (where the two domains are assumed to be different but related, the tasks are the same, and the target domain labels absent or sparse), feature-based transfer learning approaches have been successful. These assume a shared or abstract domain space to construct a mapping between a source and target domain. This shared domain space has to be carefully constructed in a task-specific way and is therefore often hand-crafted. In this paper we devise an efficient algorithm for automatic shared domain space construction, considering the task of behaviour classification of house inhabitants by sensor measurements.

1 Introduction

The ability to re-use knowledge gained at a particular domain or task in another (related) domain or task is one of the most defining charateristics of human intelligence. Yet the field of machine learning has traditionally constrained itself to the assumption that a particular learning algorithm has to be trained with data specific to the domain of the task, i.e. with the same features and equal distributions. Transfer learning is an emerging paradigm within the field of machine learning that aims to surmount this issue by attempting to transfer gained knowledge (structures) to the target task.

For this project we will be concerned with feature-based transfer. That is, we want to extract shared abstract features from the source domains to enrich the target domain, which in turn should improve classification accuracy. For any feature-based transfer approach, the specific features extracted will often be domain-specific.

1.1 Transfer learning

1.1.1 Type

The type of task or domain difference that one attempts to surmount by transferring knowledge can take several forms [6]. Below we give an overview:

1. Source and target domains are the same; source and target tasks are the same (Traditional machine learning, no transfer learning / base case)

2. Source and target domains are the same; source and target tasks differ (Inductive transfer learning)

3. Source and target domains differ; source and target tasks are the same (Transductive transfer learning)

4. Source and target domains differ; source and target tasks differ (Unsupervised transfer learning)

In this paper we focus on an experimental setting that falls into category 3, transductive transfer learning.

1.1.2 Level

With regard to the level of transfer, Davis and Domingos [3] distinguish shallow and deep types of transfer. These are defined as follows:

1. Shallow transfer assumes that the instances are from the same domain, i.e. have shared features and class values, but have different distributions.

2. Deep transfer assumes that instances are from entirely different domains - have different features and class values.

In this paper we focus on an experimental setting that falls between category 1 and 2, which will be further explained in Section 3.

1.1.3 Abstract knowledge type

In the case of transductive transfer learning, the approach is to transfer knowledge from similar, available domains (henceforth called "source domains") to the classification task on the target domain. This only works if the transferred data is general and informative enough. If this is not the case a phenomenon called "negative transfer" might appear, meaning that performance is actually harmed by the transfer of knowledge. For generalizability, one would want to abstract the knowledge gained from the source domains.

The following types of abstract "knowledge" that can be transfered can be distinguished [6]:

- Instance transfer. Re-weighting the labeled source domain data and add these instances to the training data in the target domain.
- Parameter transfer. In this case a prior distribution over parameters which need to be learned in the target task is transferred. The parameters describing the distribution that is transferred are called hyper-parameters. This distribution is used as a bias in the classification task for the target domain, as is done in the same domain we are considering in [5].
- Feature representation transfer. Abstract domain features (that reduce the differences between the source domains and target domain, and then enrich or replace the target domain) are transferred.
- Relational knowledge. Relationships within the source domain are transferred to the target domain. It is similar to feature representation transfer except for focussing on relational features in the source data.

The choice of the type of transferred knowledge is a task- and/or domain-specific one. In our case we transfer a particular type of knowledge both relational and feature based as further explained in Section 3.

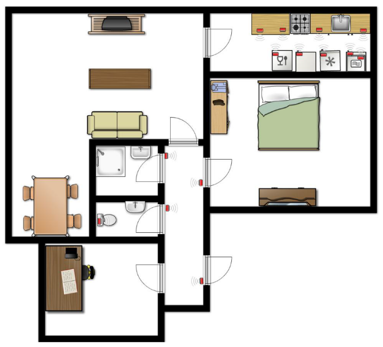
2 Problem description

In the particular transfer learning task that we focus on we have multiple different but similar domains (a set of data instances) for which we want to train the same type of classifier. The classification task and the class labels are the same for each domain. As such we have a transductive transfer task.

The target task at hand in our research is the classification of human activities such as cooking, sleeping and bathing in a home setting. For this behavior recognition task a number of sensors are placed in different homes of people. The resulting task is to be able to predict the ongoing activity based on a pattern of sensor firings. The houses are regarded as the different domains in this transfer learning task. It is clear that these domains are related. However, there are two main differences between houses:

- Across houses the sensor networks are different. This is due to the different layouts in houses (which e.g. causes sensors to be further apart, affects in-between firing times, etc.), differences in the number sensors that are placed (e.g. the number of kitchen cabinets that have a sensor) and differences in the types of sensors which are placed (e.g. sensors for light switches might be present in one house but not in others). For an example of the layout of two houses and the location of the placed sensors, see Figure 1.
- The behavioral patterns of inhabitants are different. An example of such a difference is the pace at which the inhabitant performs activities, which can be very different between the elderly and the young. Another difference is the possible discrepancy in the daily routine of inhabitants. Somebody with a night-time job will have a very different behavioral pattern compared to somebody with a day-time job.

Throughout this project we will assume that the behavioral patterns of habitants are more or less the same. We also assume that sensor activity data can be a sufficient indicator of activity class.


(a) House A

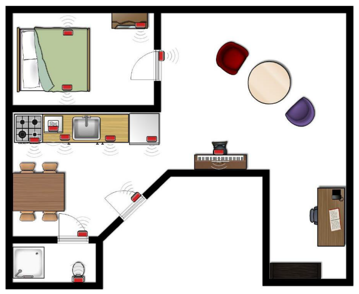

(b) House B

Figure 1: Floorplans of two of the houses indicating where sensors were placed.

3 Previous work

Our work heavily builds upon the work by Bayeva [1] and Bonenkamp et al. [2]. Bayeva [1] in turn is inspired by the work of Kasteren et al. [5]. All focus on the same task as we do, recognition of activities of daily life, using the same data set as we initally did.

Bayeva [1] focusses on relational knowledge transfer, but assumes the differences in sensor networks, as described in Section 2, to be solved. She uses a handcrafted shared feature space between sensors of different houses. Bonenkamp et al. [2] focus on this exact problem: finding a mapping between sensors of different houses. If sensor a maps to b (or a group of sensors maps to another group of sensors), this can also be seen as a and b both mapping to c. Using this intuition, such a mapping can be used to create a shared feature space between sensors of different houses.

3.1 Transfer: Abstract relational feature framework

3.1.1 Abstract knowledge type

In relational knowledge transfer, it traditionally is assumed that the relationships between the data instances are in non-i.d.d., relational form and that these relations are what has to be transferred.

In our classification case however, we would intuitively assume that the sensor firings are non-i.d.d. (related in some way) while the activities are assumed to be i.d.d. (same type of distribution and not related).

That means , we assume: instances are i.i.d., but original features within each data instance are not i.d.d. (related in some way). One therefore would want to transfer what Bayeva [1] calls "abstract relational knowledge" - knowledge that captures relationships between the original features within each individual instance.

Indeed, this is what Bayeva [1] does and we copy her approach.

3.1.2 Transfer type

This abstract relational feature approach lies between a shallow and deep transfer approach. In fact, this approach does not enforce a hard-wired choice with regard to this level of transfer. This is because the abstract relational features are generated from both the target and source domain. Thus, the right level of transfer is dependent on the difference between the source and target domains, and should automatically be determined based on the amount of overlap between the source and target domains. Little overlap would result in deep transfer, but much overlap would result in shallow transfer (because the features are abstracted just as much as they differ between the domains).

3.1.3 Implementation details

In order to extract abstract relational features the data is first expressed in first order logic. Frequent patterns in the data are then extracted per class, using a modified FP-growth-based subgroup discovery algorithm called RFE ([1, pp. 24]). This algorithm is a fast way to extract such patterns because it uses a tree-like structure to store the patterns. The resulting patterns are filtered using measures such as indicativeness of class and generality. The remaining patterns are the abstract relational features. The data from the source houses and the training data from the target house are then represented in terms of the presence of these features. This data, in turn, can be used to train an SVM classifier. For more details, see [1].

3.2 Original feature construction

We use the exact method described above for the transfer learning part of our work. However, instead of using a handcrafted shared feature space, we will use the feature space that results from automatic mapping. In Section 4 the method proposed in [2] and, in case they are present, our changes and/or additions will be discussed in mre detail.

4 Method

We have integrated the two previous approaches from Bayeva [1] andBonenkamp et al. [2]. This results in a pipeline consisting of the following steps:

1. *Clustering* Grouping sensors in a meaningful way in order to reduce the differences in the number and type of sensors between houses. The clusters are referred to as *meta-features*
2. *Mapping* Mapping the clusters from the source houses to the target house in order to build a shared feature space for transfer learning. These shared features are referred to as *meta-meta-features*
3. *Transfer learning:*
   (a) Extracting the abstract relational features from the data from source houses and training data from the target house, and representing this data in terms of those features.
   (b) Train an SVM classifier on the new representation of the data.

An overview of this pipeline can be found in Figure 2. The performance of the classifier can be evaluated on the test data from the target house.

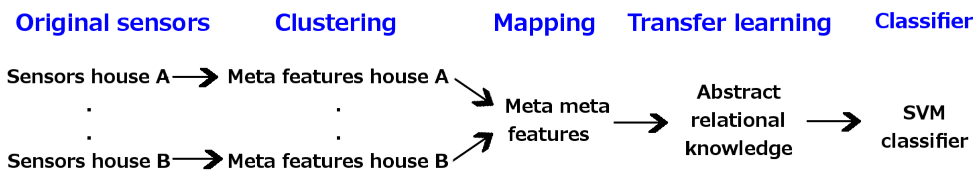


Figure 2: Overview of the transfer learning pipeline

4.1 Data

Bayeva [1] , Bonenkamp et al. [2] as well as Kasteren et al. [5] used data from just three houses. We have found another data set of two houses [1] and merged in into the original data set. We will call the houses form the original data set houses A,B and C, and the houses originally fetched from the other data set houses D and E. Because houses D and E originally

[1]http://courses.media.mit.edu/2004fall/mas622j/04.projects/home/

contained different activity labels we have mapped these labels. However house D and E still contain un-mappable activity labels not present in house A, B or C. However, this should theoretically not pose a problem.

4.1.1 Database

Along with our implementation we aimed to provide a unified behaviour recognition database which can be easily used and further extended. The specification of this unified format is also available in order to add more data from other houses in the future.

We provide labeled sensor data from two different sources [5, 7], accounting for five houses in total. Each house is inhabited by a single person. Table 1 provides a general overview on the properties of this data.

	House A	House B	House C	House D	House E
Rooms	7	3	8	5	8
Logged days	25	13	18	14	14
Sensors (all)	14	23	21	77	84
Sensors (named)	13	17	20	75	69
Sensor firings	1229	19075	22700	2772	1962
Activities (count)	8	8	8	35	35
Activities (instances)	259	112	234	295	208

Table 1: Unified database overview.

4.2 Clustering

We define a *meta-feature* as a set of sensors within one house sharing a common property which allows them to collectively abstract their behaviours from the layout of the house, while retaining useful information for the activity recognition task.

We will follow the intuition presented in [2], where sensors were grouped by their physical location. Sensors close to each other have more chances of being indicative of the same activity. This physical distance is not directly available in the data we are using, but it can be approximated by the difference in activation time between sensors: far away sensors will never fire shortly after each other.

In order to group sensors according to this intuition we use β-profiles. These are matrix-like profiles calculated on a per-house basis. Each index represents a different sensor within the house, and then each cell $c(s_1, s_2)$

contains the number of times sensor s_2 fires after sensor s_1 within a time span of length β. These profiles are thus parametrized on β. This information can be later used to determine which groups of sensors usually fire together.

Given a threshold α and the β-profiles for all houses, the meta-feature construction algorithm will iteratively merge two sensor groups S_1 and S_2 within the same house if two sensors $s_1 \in S_1$ and $s_2 \in S_2$ exist such that $c(s_1, s_2) + c(s_2, s_1) > \alpha$. Initially, each sensor will be contained within its own singleton sensor group. When the algorithm can no longer proceed, the resulting groups of sensors represent the final meta-features.

It must be noted here that the values for the parameters α and β are handmade and different per house. Appendix B contains some information about the properties of the resulting set of meta-features for several combinations of choices of α and β. An example will be discussed in Section 6.

A different option for clustering that we included is the relative α. When this method is used the resulting α is different for each sensor. It resulting α is relative-α% from the number of times the sensor fires in the available data.

4.3 Mapping

Once *meta-features* have been constructed for all houses, the shared feature space is constructed by means of mapping the *meta-features* from each source house to the target house. All clusters that map to the same target cluster are then considered to be a shared feature, i.e. are considered 'the same'. The shared features that result are the *meta-meta-features*.

We have used the procedure for cluster mapping proposed by [2], which is also verified with good results in [4]. We made a small adjustment which will be explained and motivated below. The procedure itself is as follows:

1. Start with all target clusters and source clusters as candidates.
2. Match each target cluster to the source cluster that minimizes the cluster distance.
3. Keep the match that has the smallest cluster distance as a definite mapping and remove the target and source cluster involved from the candidates.
4. Repeat step 2 and 3 until there are no more source or target clusters.

5. If source clusters remain after this procedure, each of these source clusters is mapped to the target cluster that minimizes the cluster distance.

The difference between this implementation and the one proposed by Bonenkamp et al. is the fact that remaining source clusters are mapped to target clusters instead of the other way around. The target clusters might be more refined (more clusters) or less refined (fewer clusters) compared to the source house at hand. If it is more refined, we wish to retain this since other source houses might have more refinement and we wish to keep the level of detail. If it is less refined we prefer to lower the refinement of the source house, since it is the target house that we wish to learn about.

The cluster distance referred to is defined as proposed by Bonenkamp et al. [2]. For this distance the individual profiles of the sensors are preserved. The cluster distance is then defined as the average over the best match (i.e. minimal) distances between the sensors in the two clusters. For pseudo-code of this procedure see [2, pp, 9].

The distance between sensors is defined as the difference between their profiles. Two profiles are provided in our framework:

- Sensor profile
- Relational profile

These profiles are implemented in multiple ways, which will be explained further in Section 4.4 and 4.5. The used distance metric depends on the type of profiles that is used. The distance between two sensors is simply the average of the two profile distances which have been made sure to have the same scale.

4.4 Sensor profiles

Sensor profiles model the behaviour of a particular sensor throughout a "typical" day. To build these, data from several days is merged and later averaged. We would then expect that by comparing the resulting profiles of two sensors we should be able to estimate the extent to which they exhibit similar behaviours.

Sensor profiles are implemented in [2] as a mixture of Gaussians over the starting times and the durations of the firings of the modelled sensor. This approach, however, presents several disadvantages. Firstly, it is difficult to calculate the optimal number of components a mixture of Gaussians should

use to model a sensor. Secondly, similarity comparisons between sensor profiles become computationally expensive when more than one Gaussian component is present. Finally, sensors differ greatly in their behaviours, which are usually rather complex, so modelling them as normal distributions might not be the most appropriate choice. When presenting the results in [2], the authors even state that they have the best performance according to the used measure when using a single Gaussian. This option to model a sensor using a single normal distribution is provided in our framework.
For the reasons mentioned above, we decided to implement sensor profiles as discrete histograms as well. These histograms model the normalized firing frequency of a sensor over the starting time within the day and the duration of the firing. In order to improve the usefulness of the sensor profile, the temporal resolution of the histogram can be reduced, thus becoming parametrized on the number of "bins" available for each dimension:

- In our implementation, each day is divided into "bins" of 2-hours length each. Therefore, there are only 12 possible "bins" for the starting time of firings within a histogram.

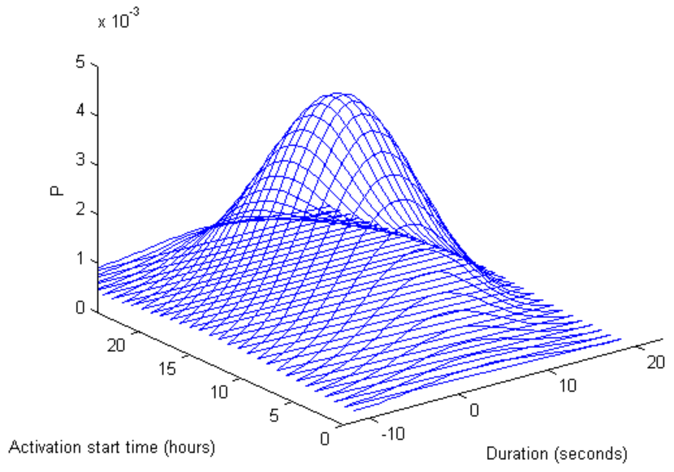
- Durations of firings can range from one second to several hours or even days. The long firings are more rare than the short ones and we do not want to loose descriptive power of short firings by putting them all in the same lowest bin. Therefore we use a logarithmic scale for firing duration. Given a maximum informative firing duration, M, and the number of desired "bins", n, our implementation automatically calculates a suitable logarithmic mapping. Formally, when using a mapping of the form $bin(x) = \lfloor \log_b \min\{x, M\} \rfloor$, b can be shown to be:

$$b = e^{\left(\frac{\ln(M+1)}{n}\right)}$$

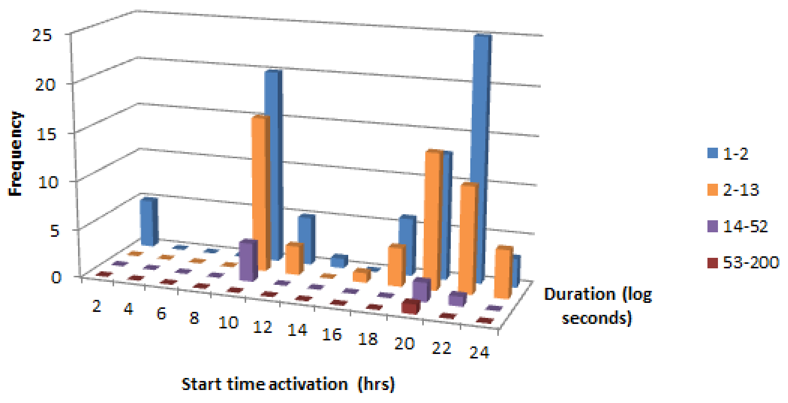
  In our implementation, we chose $M = 200$ seconds and $n = 5$.

An example of this representation can be found in Figure 3. As it can be seen, this representation is able to model the fact that the fridge is used in the morning and in the evening, as opposed to the normal distribution.

Finally, a distance measure to compare sensor profiles is needed. For the normal distribution the KL-divergence is used. With regard to the histograms we initially considered using the *Bhattacharyya distance*, as suggested in [2]. This metric is formally defined as follows:



(a) Representation as a normal distribution



(b) Representation as a histogram

Figure 3: Sensor profile using two different methods for the fridge sensor from house A.

$$dist(s_1, s_2) = -\ln \sum_i \sum_j \sqrt{s_1(i, j) \cdot s_2(i, j)}$$

However, we did not find this metric appropiate for a couple of reasons. Firstly, its possible range of values is, by definition, unbounded. This means this measure cannot be added to or weighted with other distance measures of a different kind, which is a property we would like to exploit in later stages of our work pipeline. Secondly, whenever one “bin” within a histogram equals to zero, the distance of that “bin” to any other will always be zero by definition, which is often counterintuitive. Due to this fact, and since in practice a majority of “bins” contain the value zero, using this measure results in an appreciable loss of precision.

In order to avoid these shortcomings, we decided to design a more sensible metric for this task. In our implementation, we use a distance measure called sum of squared differences, which is defined as follows:

$$dist(s_1,\, s_2) = \sqrt{\frac{\sum_i \sum_j \left(s_1(i,\, j) - s_2(i,\, j)\right)^2}{2}}$$

Since the histograms are normalized in our implementation, it can be shown that the values for this metric are bounded in the range [0, 1].

4.5 Relative profiles

Relative profiles model the temporal difference in activation times of a given pair of sensors. This information can then be used to determine which sensors fire shortly after each other, or which pairs of sensors share a similar joint firing pattern. The intuition behind this (as described by Bonenkamp et al. [2]) is that if sensor a and b have a similar relational profile to α and β, then a might map to α and b to β. However, when constructing the meta-meta-features, we need the difference of the relative profile between two sensors. If the distance of interest is between a and α, the distance between relational profiles is defined as:

$$\min_{b,\,\beta}\; dist\,(R_{a,\,b},\, R_{\alpha,\,\beta})$$

Relative profiles are modelled by Bonenkamp et al. using a normal distribution built from the set of differences in activation times of both sensors. We decided to use this idea while contributing on some details.

Firstly, it is important to decide on how to calculate the differences in activation times between firings of both sensors. Simply using all possible combinations of pairs of firings is unpractical, since it yields distributions with large variances in their data, and these are not very informative. We decided to match a fixed firing of the source sensor only with the firing of the target sensor which minimizes the temporal distance. Although this heuristic is not exact, it is easy to calculate and avoids the more complex problem of matching two sets of firings having different cardinalities.

Furthermore, the metric originally used by [2] to compare relative profiles was the *Kullback–Leibler divergence*, which measures the amount of information lost when one probability distribution is used to approximate another. This, again, results in a metric having potentially unbounded values, and thus cannot be aggregated to other distance evidence.

We instead used the amount of overlap between the two normal distributions which model the relative profiles that will be compared. This overlap can be shown to yield values within the range [0, 1]. It is formally defined as follows:

$$dist(R_1,\, R_2) = \int_{-\infty}^{+\infty} \min\left\{R_1(x),\, R_2(x)\right\} dx$$

This value cannot be easily calculated as it is expressed above, and our implementation calculates it in a more indirect way. We first solve the quadratic equation on x resulting from the equality $R_1(x) = R_2(x)$, which can return up to two solutions: m_1 and m_2 (where $m_1 \leq m_2$). We then rephrase the problem in a more computationally-friendly way. For instance:

$$\begin{aligned} dist(R_1,\, R_2) = \min\left\{\int_{-\infty}^{m_1} R_1(x)\, dx,\, \int_{-\infty}^{m_1} R_2(x)\, dx\right\} + \\ + \min\left\{\int_{m_1}^{m_2} R_1(x)\, dx,\, \int_{m_1}^{m_2} R_2(x)\, dx\right\} + \\ + \min\left\{\int_{m_2}^{+\infty} R_1(x)\, dx,\, \int_{m_2}^{+\infty} R_2(x)\, dx\right\} \end{aligned}$$

At this point, the definite integral over the distribution probability of a normal distribution can be calculated using approximations of the *error function.*

5 Results

Here we will described our experiments. We will first give an overview of meta-feature building outcomes to get an intuition behind this. Then we will compare results for different parameter settings of our automatic meta-meta-feature creation algorithm. After that we will compare automatic meta-meta-feature creation with the best found parameter setting with the base cases of using handcrafted meta-features, handcrafted meta-meta-features, and no transfer at all. Finally we will show some more experiments that may shine extra light on the outcomes.

5.1 Setup details

For all experiments in this section in which classifier accuracy is reported we mean with "classifier" an SVM classifier, for which we used libSVM [2], with the default setting of using a radial basis function kernel. The rescaling of the input values as usually done before training SVMs is also done (all values lie between 0 and 1).

The approach of evaluating classifier performance is done by picking a number of days which refer to the number of days on which the target house is trained. For such a number-of-days-setting, a random sensor and activity data sample is drawn from the target house domain data, where one instance in this sample is one day, consisting of sensor and activity data for "number-of-days" days. This randomly drawn sample is split into multiple test and train set using the leave-one-out approach (e.g. 4 sets for 4 days). The number of times a random sample is drawn (and its leave-one-out sets are added to the train and test instances) we call the "number of train/test-set samples".

In cases where we report "average accuracy" we mean with this: the accuracy, averaged over multiple samples of train/test-sets. Additionally, when we report the average accuracy for several houses, the accuracy is of course also averaged over all these houses.

When we show a plot for e.g. "House A"we mean with this that house A was considered the target house in that experiment and all other houses as the source. When we show a plot for e.g. "House A, B, C"we mean with this that house A, B as well as C was considered the target house in that experiment, with the source houses being, respectively, {B,C},{A,C},{A,B}.

5.2 Meta-feature building

Before looking into the results on transfer learning, we would like to show a peculiar difficulty with regard to re-implementing the clustering method described by Bonenkamp et al. Table 2 shows some statistics for house C after clustering for different values for α and β. It is not clear from the provided results in [2] how the clustering process depends on α and β, and how this might scale to houses with a very different number of sensors. It

[2]`http://www.csie.ntu.edu.tw/~cjlin/libsvm/`

can be seen in the statistics that for house C it is not possible to generate more than two non-unary clusters, and that, depending on the parameters, only these clusters vary in size. This pattern emerges for all houses, as can be checked in Appendix B which contains these statistics for all houses. In house C this needs not to be a problem. However, when the number of sensors becomes larger such as in house D and E, the large number of unary clusters causes a very skewed distribution over the number of clusters per house. Since we wish to make shared features, this is not desirable at all.

Beta	**Statistic**	**Alpha**		
		1	3	5
3	Nr clusters	7	11	15
	Nr nu-clusters	2	1	2
	Avg size nu-clusters	8.0	11.0	4.0
6	Nr clusters	6	9	9
	Nr nu-clusters	2	2	2
	Avg size nu-clusters	8.5	7.0	7.0
9	Nr clusters	2	6	9
	Nr nu-clusters	2	2	2
	Avg size nu-clusters	10.5	8.5	7.0
12	Nr clusters	1	4	8
	Nr nu-clusters	1	2	2
	Avg size nu-clusters	21.0	9.5	7.5

Table 2: Cluster statistics for house C after clustering with different values for α and β. *Nu-clusters* stand for *non-unary-clusters.*

5.3 Automatic meta-meta-feature construction methods

The first experiment we conducted aims to see which set of parameters when constructing the meta-meta-features has the best result during transfer learning. All results for this and further experiments result from applying the leave-one-out procedure multiple times. Each time the required number of labeled days is sampled from the available data. Here, we use 50 samples.

The different meta-meta-feature building settings consist of all combinations of the following options:

- *CT_ABS vs CT_REL*: using absolute or relative alpha.
- *PR_BOTH vs PR_SP*: using both the relative and sensor profile or using only the sensor profile.

- *SSE vs KL*: using a histogram representation and therefore the sum of squared differences metric for the sensor profile or using the normal distribution representation and therefore the KL-divergence metric.

In more detail, we set the experiments setting as in Table 3.

Setting	**Value**
alphaAllHouses	10 10 10 4 4
betaAllHouses	9 9 9 6 9
relativeAlpha	0.01
bin_width_start_time	two hours
nr_bins_duration	5
max_length_duration	200
nr_train/test_samples	50
additional sample constraint	(nr_train/test_samples * nr_days_in_sample) $\leq$ 300

Table 3: Settings of the automatic meta-meta-feture construction methods experiments.

If a setting is not mentioned, it is the same as in Bayeva [1].

The settings related to the sensor modeling in Table 3 (also see Section 4.4) may require some further explanation. The list "alphaAllHouses" lists the absolute α settings in increasing house letter order (e.g. the third value ipertains to house C). Of course the absolute α setting only pertains to the case of *CT_ABS* and not *CT_REL*, and vice versa for the relative α setting. The additional sample constraint was mainly provided by us to confine the total run time of the transfer learning algorithm. The parameter bin_width_start_time is the size of the bins for discretization of the activation start time dimension, and the parameter nr_bins_duration pertains to the amount of bins for the duration dimension. The parameter max_length_duration pertains to the maximum informative firing length: lengths above this value will simply be regarded as being this value.

Figure 4 contains the results of the SVM classifier after transfer learning for different settings in the automatic meta-meta-feature construction for each house.

When averaged over all houses the best combination of settings seems to be using a relative alpha, both profiles and representing the sensor profiles as a normal distribution and use KL-divergence as a distance metric. However, the differences between the different combinations of settings are small. Due to space constrains we do not include the results seperate for each house here, these are available in Appendix A.1. When looking at the seperate results

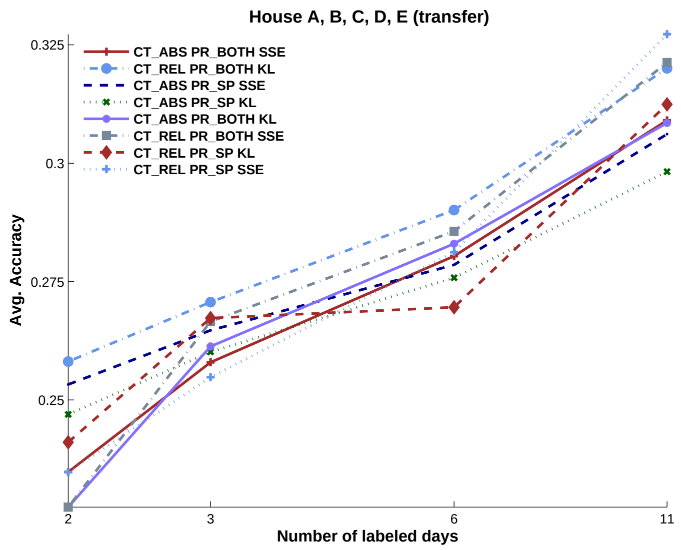


Figure 4: The accuracy of the SVM classifier after transfer learning for different settings in the automatic meta-meta-feature construction averaged over all houses.

it also becomes clear that this particular setting is not best for all houses. Still, we will use this configuration for the next experiment.

5.4 Comparison with base cases

The second experiment consists of comparing the best performing setting for automatic meta-meta-feature construction compared to a setting where the meta-features are constructed by hand and the mapping is automatic, a setting where the meta-meta-features are handcrafted (as in [1]) and finally, the no-transfer case (for which the choice of meta-meta-features does not matter, and we use the handcrafted ones) . When applicable, we used the same settings as in Table 3.The results for this experiment can be seen in Figure 5.

The four experiment settings are the following:

- *AUTO CT_REL PR_BOTH KL TRANSFER*: This is the automatic meta-meta-feature construction setting with a relative clustering setting (relative α), using both the sensor and relational profile, the KL-divergence distance measure. It was the setting deemed best in Section 5.3. *TRANSFER* indicated we use the setting of transferring the knowledge.

- *HC_MF PR_BOTH KL TRANSFER*: Handcrafted meta-features (handcrafted by us). This means only mapping, i.e. the meta-meta-feature construction, is done automatically with using both the sensor and relational profile, and as a disance measure the KL-divergence.
- *HC_MMF TRANSFER*: This is the setting of using handcrafted meta-meta-features, i.e.
- *HC_MMF TNORANSFER*: Same as above, but indicates we use the setting of not transferring any knowledge. In this case the meta-meta-feature used don't matter.

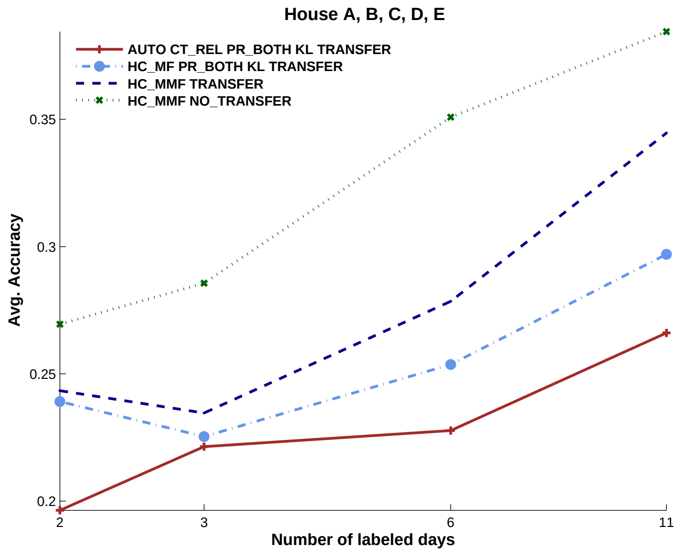


Figure 5: The accuracy of the SVM classifier comparing transfer learning with different amounts of automatization in the meta-meta-feature construction and the no transfer setting. This comparison is an average over all houses.

It is clear that the average accuracy is highest for the non-transfer case. For any number of labeled days from the target house used for training, there is negative transfer, i.e. using knowledge from other domains harms the performance. Using automatically constructed meta-meta-features also harms performance. This harm is caused partially by the mapping procedure and partially due to clustering. The performance is harmed when using handcrafted meta-features which are mapped automatically, although not as much as when the entire procedure is automatic.

5.5 Investigating causes

The results in Section 5.4 are not what we expected. They are worse than the results reported by Bayeva [1, pp. 40] on comparable measures.

5.5.1 Fewer houses

We have conducted the same experiment using only house A, B and C, since Bayeva [1] only used data from these houses. Furthermore, the amount of activity labels is larger for houses D and E, which come from a different source, than for houses A,B and C. We suspected that this could bias the subgroup discovery algorithm Bayeva [1] and we use.

The results when only using these three houses can be found in Figure 6. Even though these results are strictly better for all conditions compared to the results when using all houses, transfer learning never outperforms the case where there is no transfer learning, as is reported by Bayeva [1].

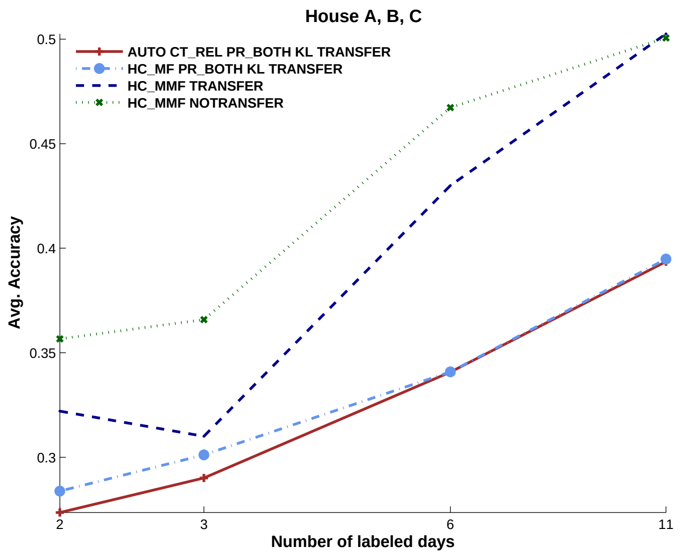
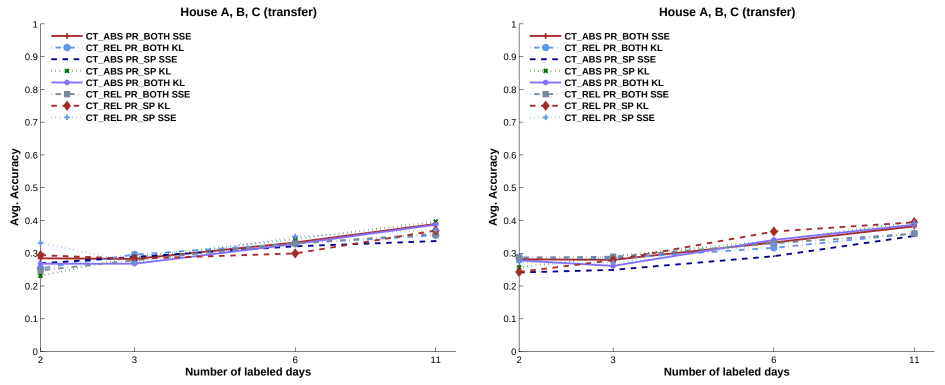


Figure 6: The accuracy of the SVM classifier comparing transfer learning with different amounts of automatization in the meta-meta-feature construction and the no transfer setting. This comparison is an average over houses A, B and C.

5.5.2 More train/test-samples

We furthermore checked if he results of Section 5.3 would improve using more train/test data samples per number of days. We set nr_train/test_samples to 100 and the additional sample constraint to (sample-size * nr_days_in_sample) $\leq$ 500.

This improvement turned out not to be the case, as can be seen in Figure 7. Here we compare it with the results already visualized in Figure 4 but now with a rescaled vertical axis in Figure 7(a) to better visualize the (lack of) difference with relation to the setting of grabbing 100 instances in Figure 7(b).



(a) Max. 50 train/test-samples (b) Max. 100 train/test-samples

Figure 7: Comparison of max. train/test samples number for automatic meta-meta-feature construction. Metric is SVM classifier accuracy.

5.5.3 Replicating original results

As a final sanity check and attempt to replicate the results of Bayeva [1] we ran our algorithm for the handcrafted meta-meta-feaures case on both transfer and no-transfer settings, for house A,B and C only, using a maximum 100 train/test-samples without limitation. That is, we set nr_train/test_samples to 100 and the additional sample constraint was removed. In addition we also tested our best automatic meta-meta-feature construction setting as given in Section 5.3 under this case.

Here we do not report the SVM classifier accuracy average over all houses ("Average accuracy") as in the previous results. Instead, to in accordance with the experiments of Bayeva [1] we report the "Average average accuracy"(Bayeva [1, pp. 38]) which is in our case also averaged over houses and defined as:

$$Average\ average\ accuracy = \sum_{h=1}^{H} \left(\frac{\sum_{c=1}^{C_h} TP_{h_c}}{\sum_{c=1}^{C_h} SampleSize_{h_c}} \right)$$

where H is the total number of houses; C_h the total number of activity classes per house h, TP_{h_c} refers to the correctly classified examples ("true positives") for class c for house h , and $SampleSize_{h_c}$ is the number of action class instances for class c for house h .

This is done to account for differences in class distributions, and will result in treating misclassifying e.g. 1 out of a total of 2 instances for an activity class as less severe than misclassifying 50 out of 100 class activity instances.

However we can see in Figure 8 that this does not improve our results. In Figure 9(b) we have plotted the results for house A and house B separately to show that the house instance seems to matter a lot.

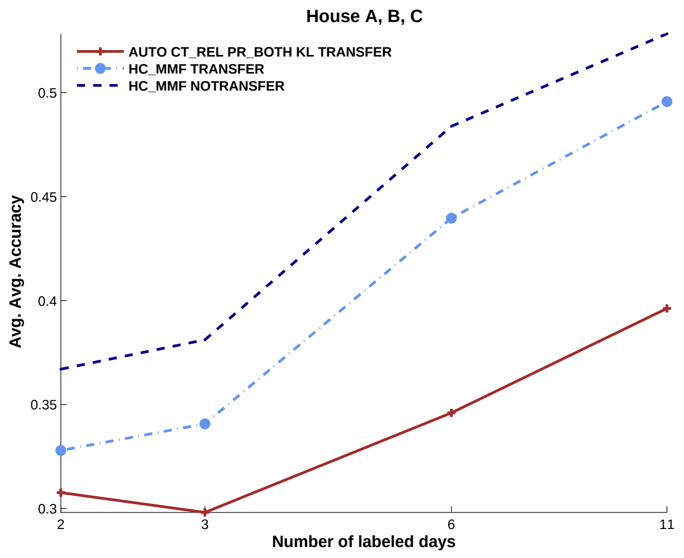
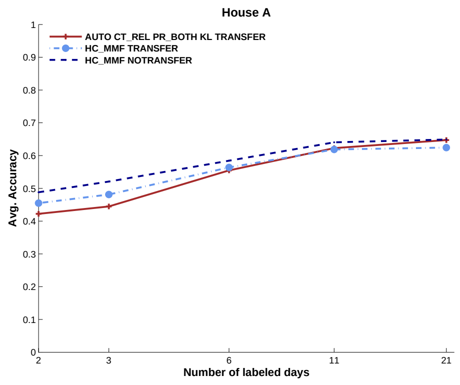
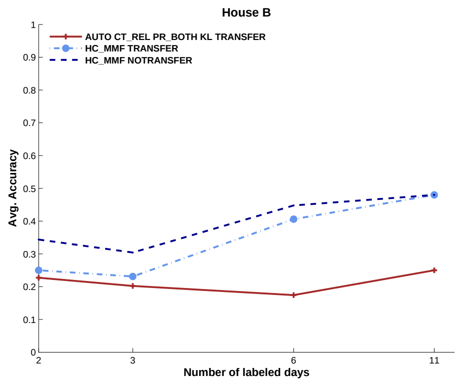


Figure 8: Averaged house results for max. 100 train/test-samples per nr. of days without additional sample size limits. Metric is the average over classes of the SVM classifier average over houses accuracy. (Same setting as in Bayeva [1, pp. 40].)



(a) House A



(b) House B

Figure 9: Averaged house results for or max. 100 train/test-samples per nr. of days without additional sample size limits. Metric is the average over classes of the SVM classifier average over houses accuracy. (Same setting as in Bayeva [1, pp. 40].)

6 Discussion

The results from Section 5 seem to warrant a more in-depth investigation. Due to the time scope limitation of our project, we will cofine to giving some pointers in this section.

6.1 Meta-meta-feature building

Constructing the meta-meta-features automatically is a large part of our pipeline. We have two remarks with regard to this procedure.

6.1.1 Alpha and beta parameters

The method that is currently used to cluster sensors does not seem to perform well. Besides the fact that performance using automatically constructed clusters is worse than using handcrafted clusters, the whole assumption on which they are grouped seems suboptimal. Ideally one would want to cluster sensors that have a similar function, or are indicative of the same activity. Currently the sensors are grouped based on (an approximation of) physical closeness.

6.1.2 Relational profile

The number of relational profile distances that need to be calculated is very large, which increases computation time in an extreme manner. When mapping a source house with N_S sensors to a targethouse with N_T sensors, the number of relational profile distances that needs to be calculated is $N_S^2 \cdot N_T^2$. However, when looking at the results for different settings split per house (appendix A.1) the only setting that always outperforms the alternative setting for all houses, is using both the relative and sensor profile. The information captured in the relational profile therefore seems very usefull, but maybe a computationally cheaper alternative can be found.

6.2 Transfer results

If we exclude the possibility of bugs in our implementation, which we will do here because we have investigated all the code thoroughly, the failure for the transfer setting to improve accuracy for the non-transfer setting seems to be a case of negative transfer. This is possible even in the case of using the original data (as in Figure 8) because we do not know the original transfer parameter and SVM parameter settings of [1].

The appearance of negative transfer could mean that somehow the abstract relational features do not contain information that is helpful for classification and instead causes noise that harms the performance. This can be caused by the fact that there is not enough general information present in the source domain, i.e. the related domains turn out to be insufficiently related. A different cause can be that there is general information present, but there is not enough training data in the target domain to counter the general information and making the classification task domain specific.

In our case following Bayeva [1] we replace the original target train data completely with abstract features which may be too ambitious when having a sparse data. In cases of sparse data it might be better to use abstrct knowledge as a bias.

7 Conclusion

The clustering results of Bonenkamp et al. [2] could be replicated. It failed, however, to result in good transfer learning performance, since the decrease in performance using automatically constructed meta-meta-features is considerable.

The different approaches to automatic meta-meta-feature construction do not seem to affect the final transfer learner that much. Results were not very different for different houses, and inconsistent depending on the house chosen as target house.

Unfortunately we have not been able to replicate the results reported by Barayeva [1], since we have only found negative transfer in all settings.

8 Future work

There are several points throughout this project which we believe can be improved and thus are a good starting point for future work. For instance, we think that the α-β clustering method can be vastly improved. Firstly, its parameters are difficult to estimate. Besides, it also seems intuitive to use the labelled data that is available in the source houses to aid the clustering process, but this is not considered at all within the current approach. Finally, it could also be interesting to implement some sort of hierarchical clustering method, in order to gain more control over the number and size of the clusters that are formed.

It might be a good option to use the relational profile distance across clusters instead of sensors, i.e. let go of individual sensor profiles with regard to the relational profile. Since there are far fewer clusters than sensors this can reduce computation time substantially. If the clusters are somehow meaningful, the intuition from the relational profile still stands. Even though the sensor profiles of a cluster might be different across inhabitants the relation to other clusters might stay the same. This could still aid in constructing good mappings.

The computational cost of assessing the impact of different approaches to meta-meta-feature construction using transfer learning are large. It would be very desirable to have a different measure to assess the meta-meta-feature construction, such that further investigation on meta-meta-feature creation would be easier to conduct as well as having a more solid theoretical background.

References

[1] Y. Bayeva. Transfer Learning with Abstract Relational Features. Master's thesis, University of Amsterdam, the Netherlands, August 2011.

[2] K. Bonenkamp, D. Chiappetta, and B. Bittner. Variable mapping for

transfer learning. Profile project AI, University of Amsterdam, the Netherlands, February 2012.

[3] J. Davis and P. Domingos. Deep transfer via second-order markov logic. In *Proceedings of the 26th annual international conference on machine learning*, pages 217–224. ACM, 2009.

[4] M. Jakobsson and N. A. Rosenberg. Clumpp: a cluster matching and permutation program for dealing with label switching and multimodality in analysis of population structure. *Bioinformatics*, 23(14):1801–1806, 2007.

[5] T. Kasteren, G. Englebienne, and B. Kröse. Transferring knowledge of activity recognition across sensor networks. In P. Floréen, A. Krüger, and M. Spasojevic, editors, *Pervasive Computing*, volume 6030 of *Lecture Notes in Computer Science*, pages 283–300. Springer Berlin Heidelberg, 2010. ISBN 978-3-642-12653-6.

[6] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Trans. on Knowl. and Data Eng.*, 22(10):1345–1359, Oct. 2010. ISSN 1041-4347.

[7] E. M. Tapia. Activity Recognition in the Home Setting Using Simple and Ubiquitous Sensors. Master's thesis, Massachusetts Institute of Technology, USA, September 2003.

Appendices

A Transfer results using all houses and 50 samples

This appendix contains the results of the SVM classifier accuracy after transfer learning for different settings. The number of train/test-instances is in all these cases set to 50. Furthermore in all these cases we are using all houses: A,B,C and D. For details about the parameter settings and naming see Section 5.

A.1 Automatic meta-meta-feature construction methods

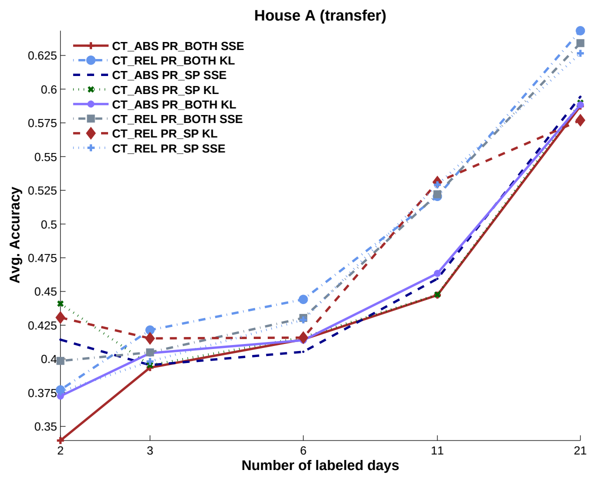


Figure 10: The accuracy of the SVM classifier after transfer learning for different settings in the automatic meta-meta-feature construction for house A

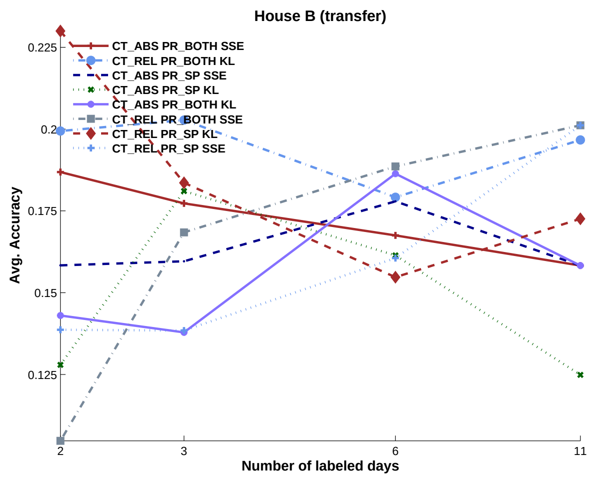


Figure 11: The accuracy of the SVM classifier after transfer learning for different settings in the automatic meta-meta-feature construction for house B

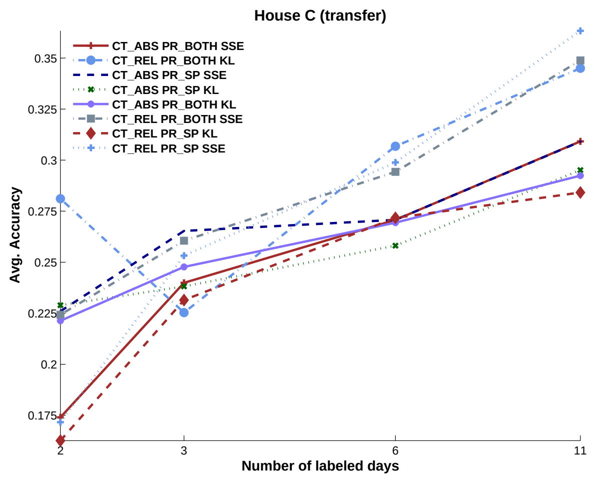


Figure 12: The accuracy of the SVM classifier after transfer learning for different settings in the automatic meta-meta-feature construction for house C

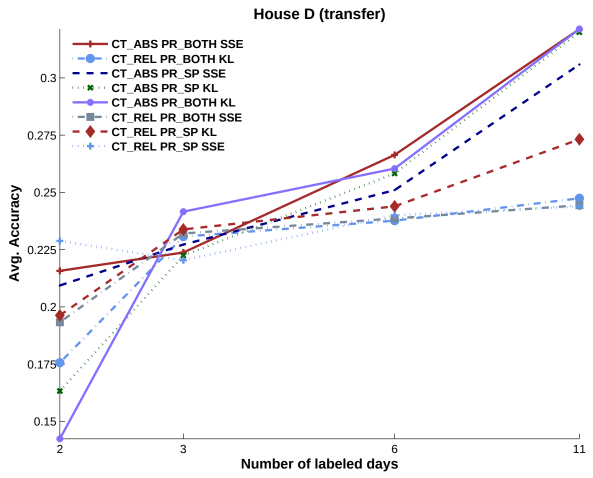


Figure 13: The accuracy of the SVM classifier after transfer learning for different settings in the automatic meta-meta-feature construction for house D

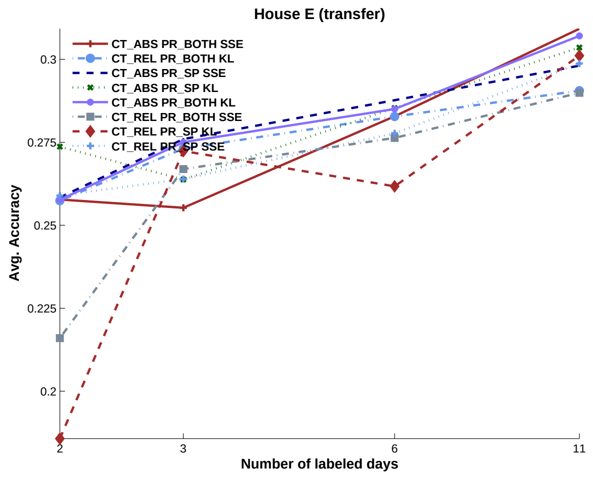


Figure 14: The accuracy of the SVM classifier after transfer learning for different settings in the automatic meta-meta-feature construction for house E

A.2 Comparison with base cases

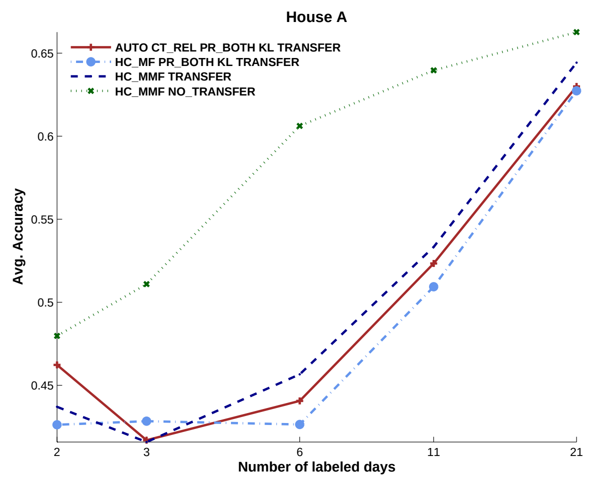


Figure 15: The accuracy of the SVM classifier for max. 50 train/test-samples per nr. of days when comparing our best automatic approach the the three base cases for house A.

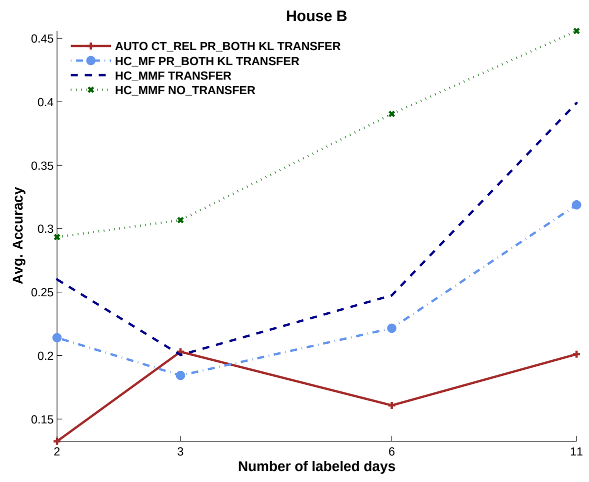


Figure 16: The accuracy of the SVM classifier for max. 50 train/test-samples per nr. of days when comparing our best automatic approach the the three base cases for house B.

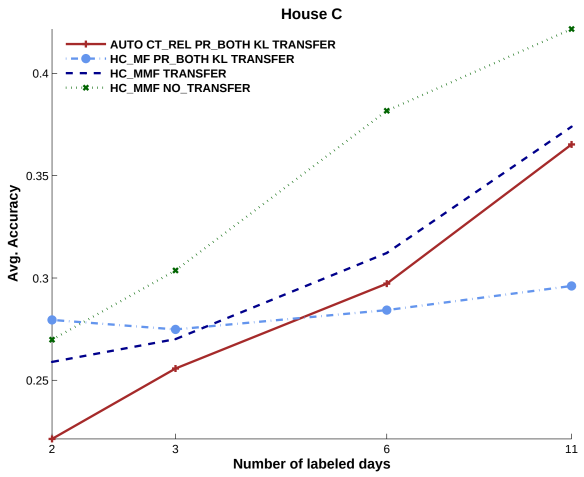


Figure 17: The accuracy of the SVM classifier for max. 50 train/test-samples per nr. of days when comparing our best automatic approach the the three base cases for house C.

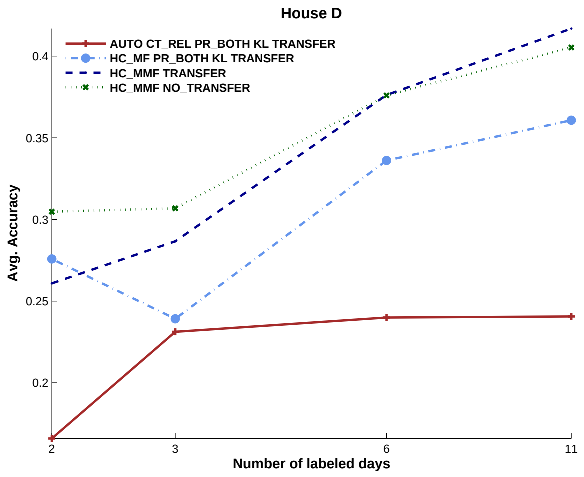


Figure 18: The accuracy of the SVM classifier for max. 50 train/test-samples per nr. of days when comparing our best automatic approach the the three base cases for house D.

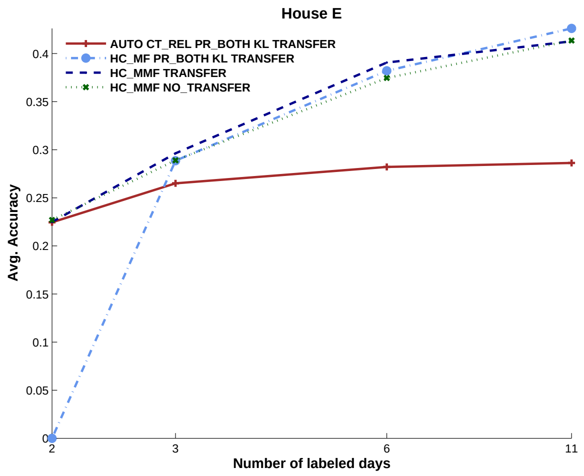


Figure 19: The accuracy of the SVM classifier for max. 50 train/test-samples per nr. of days when comparing our best automatic approach the the three base cases for house E.

B Tables for alpha-beta values

This appendix contains cluster statistics for all houses after clustering with different values for α and β. Nu-clusters are non-unary-clusters.

Beta	Statistic	Alpha						
		1	2	3	4	6	8	10
3	Nr clusters	9	9	10	10	11	12	12
	Nr nu-clusters	2	2	2	2	1	1	1
	Avg size nu-clusters	3.5	3.5	3.0	3.0	4.0	3.0	3.0
6	Nr clusters	6	6	8	8	9	9	10
	Nr nu-clusters	2	2	2	2	3	3	2
	Avg size nu-clusters	5.0	5.0	4.0	4.0	2.67	2.67	3.0
9	Nr clusters	6	6	6	6	8	8	9
	Nr nu-clusters	2	2	2	2	2	2	3
	Avg size nu-clusters	5.0	5.0	5.0	5.0	4.0	4.0	2.67
12	Nr clusters	3	5	6	6	7	8	8
	Nr nu-clusters	1	2	2	2	2	2	2
	Avg size nu-clusters	12.0	5.5	5.0	5.0	4.5	4.0	4.0
15	Nr clusters	2	5	6	6	6	6	8
	Nr nu-clusters	1	2	2	2	2	2	2
	Avg size nu-clusters	13.0	5.5	5.0	5.0	5.0	5.0	4.0

Table 4: House A

Beta	Statistic	Alpha						
		1	2	3	4	6	8	10
3	Nr clusters	1	1	2	2	2	2	4
	Nr nu-clusters	1	1	1	1	1	1	1
	Avg size nu-clusters	18.0	18.0	17.0	17.0	17.0	17.0	15.0
6	Nr clusters	1	1	1	1	2	2	2
	Nr nu-clusters	1	1	1	1	1	1	1
	Avg size nu-clusters	18.0	18.0	18.0	18.0	17.0	17.0	17.0
9	Nr clusters	1	1	1	1	1	2	2
	Nr nu-clusters	1	1	1	1	1	1	1
	Avg size nu-clusters	18.0	18.0	18.0	18.0	18.0	17.0	17.0
12	Nr clusters	1	1	1	1	1	1	1
	Nr nu-clusters	1	1	1	1	1	1	1
	Avg size nu-clusters	18.0	18.0	18.0	18.0	18.0	18.0	18.0
15	Nr clusters	1	1	1	1	1	1	1
	Nr nu-clusters	1	1	1	1	1	1	1
	Avg size nu-clusters	18.0	18.0	18.0	18.0	18.0	18.0	18.0

Table 5: House B

Beta	Statistic	Alpha						
		1	2	3	4	6	8	10
3	Nr clusters	7	8	11	13	15	16	16
	Nr nu-clusters	2	1	1	1	2	2	2
	Avg size nu-clusters	8.0	14.0	11.0	9.0	4.0	3.5	3.5
6	Nr clusters	6	6	9	9	10	14	16
	Nr nu-clusters	2	2	2	2	3	3	2
	Avg size nu-clusters	8.5	8.5	7.0	7.0	4.67	3.33	3.5
9	Nr clusters	2	5	6	7	9	9	12
	Nr nu-clusters	2	2	2	2	2	2	3
	Avg size nu-clusters	10.5	9.0	8.5	8.0	7.0	7.0	4.0
12	Nr clusters	1	2	4	7	8	8	9
	Nr nu-clusters	1	2	2	2	2	2	2
	Avg size nu-clusters	21.0	10.5	9.5	8.0	7.5	7.5	7.0
15	Nr clusters	1	2	4	6	8	8	8
	Nr nu-clusters	1	2	2	2	2	2	2
	Avg size nu-clusters	21.0	10.5	9.5	8.5	7.5	7.5	7.5

Table 6: House C

Beta	Statistic	Alpha						
		1	2	3	4	6	8	10
3	Nr clusters	30	47	50	54	58	61	62
	Nr nu-clusters	4	8	8	7	8	7	7
	Avg size nu-clusters	11.5	4.12	3.75	3.57	2.75	2.57	2.43
6	Nr clusters	21	35	39	46	57	60	60
	Nr nu-clusters	1	3	3	7	8	6	6
	Avg size nu-clusters	52.0	13.33	12.0	4.71	2.88	3.0	3.0
9	Nr clusters	16	31	33	39	48	55	57
	Nr nu-clusters	1	1	2	5	5	6	6
	Avg size nu-clusters	57.0	42.0	20.5	7.6	5.8	3.83	3.5
12	Nr clusters	15	26	29	37	44	49	54
	Nr nu-clusters	1	4	3	4	4	5	5
	Avg size nu-clusters	58.0	12.5	15.33	9.75	8.0	5.6	4.6
15	Nr clusters	12	24	27	31	41	48	54
	Nr nu-clusters	1	4	3	5	6	4	5
	Avg size nu-clusters	61.0	13.0	16.0	9.2	6.17	7.0	4.6

Table 7: House D

Beta	Statistic	Alpha						
		1	2	3	4	6	8	10
3	Nr clusters	49	53	55	63	65	66	66
	Nr nu-clusters	5	5	5	4	2	1	1
	Avg size nu-clusters	4.8	4.0	3.6	2.25	2.5	3.0	3.0
6	Nr clusters	44	46	48	56	60	63	65
	Nr nu-clusters	4	4	4	7	6	4	2
	Avg size nu-clusters	7.0	6.5	6.0	2.71	2.33	2.25	2.5
9	Nr clusters	41	42	44	52	56	61	64
	Nr nu-clusters	3	2	2	7	6	6	3
	Avg size nu-clusters	10.0	14.0	13.0	3.29	3.0	2.17	2.33
12	Nr clusters	39	41	43	48	54	57	60
	Nr nu-clusters	3	2	2	5	5	5	4
	Avg size nu-clusters	10.67	14.5	13.5	5.0	3.8	3.2	3.0
15	Nr clusters	35	37	39	45	52	56	58
	Nr nu-clusters	1	1	1	3	4	5	4
	Avg size nu-clusters	34.0	32.0	30.0	8.67	5.0	3.4	3.5

Table 8: House E