

Model Recommendation for Action Recognition

Pyry Matikainen¹ Rahul Sukthankar^{2,1} Martial Hebert¹

pmatikai@cs.cmu.edu rahuls@cs.cmu.edu hebert@ri.cmu.edu

¹The Robotics Institute, Carnegie Mellon University

²Google Research

Abstract

Simply choosing one model out of a large set of possibilities for a given vision task is a surprisingly difficult problem, especially if there is limited evaluation data with which to distinguish among models, such as when choosing the best “walk” action classifier from a large pool of classifiers tuned for different viewing angles, lighting conditions, and background clutter. In this paper we suggest that this problem of selecting a good model can be recast as a recommendation problem, where the goal is to recommend a good model for a particular task based on how well a limited probe set of models appears to perform. Through this conceptual remapping, we can bring to bear all the collaborative filtering techniques developed for consumer recommender systems (e.g., Netflix, Amazon.com). We test this hypothesis on action recognition, and find that even when every model has been directly rated on a training set, recommendation finds better selections for the corresponding test set than the best performers on the training set.

1. Introduction

Consider the following near-future scenario: John wishes to use his smartphone to solve some particular vision task — it might be face detection, or object recognition, or action classification, or any of a number of different tasks in different vision areas (in this paper we consider action recognition, but the method is general). John only has the time and patience to collect a very limited number of training samples for his task, on the order of ten samples at most. Because this is the near and not the far future, John does not have access to some super-representation of the visual world, some ‘ultimate feature’ that would let him directly train a good classifier from his ten samples. Instead, he is faced with an array of competing techniques and representations, all of which tend to produce classifiers or models that are very dataset specific and brittle to variation.

The problem faced by John is this: given a large, un-

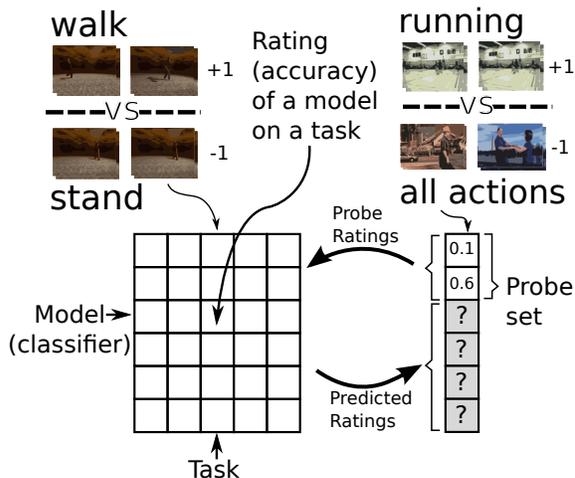


Figure 1: The goal of model recommendation is to predict the accuracy of models (classifiers) on a task based on how well a probe set of classifiers perform on that task, and a database of how well the classifiers perform on other tasks. In this way a good classifier can be selected from a large pool by only testing a small number of classifiers from the pool.

organized, heterogeneous collection of models (classifiers) trained from different data sources, how should he select a good one for his immediate task? In principle, he might just evaluate every model against his training data, but this is complicated by two issues. First, for a large library it will be impractical to evaluate all of the models on John’s task. However, the more serious issue is that evaluating a large number of models against a very small dataset will be very noisy, and the apparently best performing models are just as likely to be due to chance as to any real superiority.

The key insight of this paper is that the problem of selecting some *option* (in this case, a classifier) from a large library can be conceptually reinterpreted as the type of recommendation problem addressed by collaborative filtering methods such as those employed by Netflix to make movie recommendations (Fig. 1). Through this transformation, the

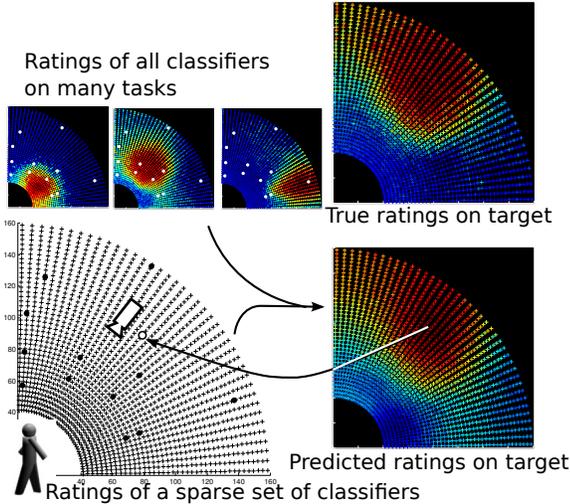


Figure 2: This figure illustrates the general concept of Fig. 1 in one specific case: view-dependent action recognizers. Given a pool of 1600 classifiers (black ‘+’s), trained to recognize an action from different viewpoints and a ratings store describing their performance on other tasks, we identify the best classifier to recognize “walk” from a particular viewpoint (white dot) by *accurately predicting* the performance of each classifier in the pool on the novel task using just 12 probes (black dots). Note that the system is *not* given meta-data about classifier locations, but must learn the correlations between classifiers.

performance ratings (accuracies) of a small subset of classifiers on a target task can be used to predict, or recommend, a good classifier from the full library in the same way that a user’s historical ratings of a small set of movies can be used to recommend unseen movies that are likely to be enjoyed.

However, because the typical domain seen in recommender systems is so vastly different, it is only right to be skeptical; what reason is there to believe that methods developed to predict the preferences of human beings will have any real traction when applied to predict the accuracies of classifiers? We experimentally demonstrate that this conceptual reinterpretation reflects an exploitable parallel, rather than being merely a cute metaphor. Because we wish to demonstrate how tight this analogy is, we apply an existing collaborative filtering algorithm *without modification*.

The metaphorical ‘near future’ scenario serves as a illustration for a formulation that can be applied to a variety of current, real-world situations. For example, it is possible to tune action recognizers for a variety of conditions, such as time of day, viewing angle, clutter, or crowdedness. However, at the time of application simply estimating these conditions may be a difficult problem, complicating the selection of the appropriately tuned classifier, especially if the amount of training data is limited. Model recommendation

sidesteps the problem of estimating the conditions by directly learning the relationships between models to recommend ones that are likely to perform well on a new task.

Furthermore, there is an interesting difference between model recommendation and typical recommender systems: in a typical recommender system, where the ratings are human preferences, the system can never do better than if a person were to directly rate every item, because the person’s preferences are ground truth. However, the same is not true for model recommendation, because the ratings (accuracies) are derived from known *training* data, but what is really wanted is to predict which classifiers are best on the withheld *testing* data. Thus, it is theoretically possible with recommendation to do better than trying every option and picking the best, and we find that this is not simply a curious theoretical possibility, but a consistent empirical effect.

2. Overview and terminology

We formulate the problem as follows (see Fig. 1): a user wishes to find a *model* that performs well on a (action recognition) *target task*, but has limited training data. The user evaluates, or *rates*, only a small subset, called the *probe set*, of the models on their task, and returns the rated performance of that small subset. The goal is to use that subset of ratings to predict the ratings of the remaining models in the pool, and return the model with the highest predicted rating.

Formally, a **task** T_j is assumed to be a forced choice classification problem. We denote a task

$$T_j = \{(x_{j1}, y_{j1}), (x_{j2}, y_{j2}), \dots\}, \quad (1)$$

where $x_{j,z}$ is the z th data sample associated with task T_j , and $y_{j,z}$ is the label corresponding to that sample. The data sample $x_{j,z}$ might be some large representation (*e.g.*, a video clip), while the target label $y_{j,z}$ is a binary value indicating the class to which a data sample belongs. Note that labels have different interpretations across tasks; in one task the binary labeling may mean “run vs. jog”, while in another it may mean “jump vs. all”.

A **model** is a classifier whose accuracy can be measured on a task (or more generally, any algorithm that can be assigned a quality rating on a task). A model can vary from having no free parameters (a pre-trained classifier) to being simply a methodology (*e.g.*, STIP+HOG+MKL) that is trained on a task’s training data. In this paper we use libraries of pre-trained classifiers that all share the same methodology (STIP+HOG3D+SVM), but which differ in their training data.

The **rating** of a model on a task is a number describing how ‘good’ that model is for that task, where higher is better. The rating of a model on a task is denoted by $R(m_i, T_j) = r_{i,j}$, and is restricted to the range $[0, 1]$. In practice, we simply use the classification accuracy of a model on a task as the corresponding rating.

2.1. A Thought Experiment

As a thought experiment as to how model recommendation might work, consider a set of models where each model is trained to recognize walking from a different viewing angle, and a set of tasks where the objective of each task is to recognize walking from a given viewing angle.

Now, for a new task with an *unknown* viewing angle, the goal is to pick the best model. Intuitively, we would expect the performance or rating of a model trained for angle θ_m on a task with target viewing angle θ_t to be inversely correlated with the difference in angles. If we let $f_m = \langle \sin(\theta_m), \cos(\theta_m) \rangle$ and $f_t = \langle \sin(\theta_t), \cos(\theta_t) \rangle$, we might expect the rating to be proportional to the dot product between the respective *factors* vectors, or $r_{m,t} \propto f_m \cdot f_t = f_m^T f_t$, with some constant offsets and scales. If this hypothesis were true, then ideally by evaluating only two models, the performance of every other model could be predicted.

A critical distinction must be noted: while every rating could ideally be predicted from only two ratings, it does not follow that every corresponding model could be reconstructed from only two models. That is to say, by rating only the 0° and 90° models on a task, the ratings of all the other models might be predicted, but no ensemble of those two models would produce a model optimized for 45° . In practice, the estimated rating for any model is noisy, so combining ratings from additional models beyond two should still improve prediction quality.

We test this scenario in the evaluation section (Sec. 5.1) and find that the hypothesis is largely correct.

3. Related work

Our work is conceptually related to domain adaptation and multi-task learning, in that the objective is to adapt or learn from the database of tasks to the target task. In domain adaptation, many methods, such as those by Gopalan et al. [10] and Saenko et al. [22], use a source domain to improve performance in a target domain by mapping samples from the source to the target. However, in the model recommendation problem the training samples for the models are not available, making this type of transfer difficult.

More closely related to our method are techniques focused on adaptation or transfer through feature selection. Blitzer et al. use a set of ‘pivot features’ [4] to perform domain adaptation, by learning the relationships between a small number of labeled pivot features and a larger number of unlabeled features in both domains. Through the pivot features they attempt to transfer models built on the unlabeled features between domains. Likewise, a meta-learning approach by Mierswa and Wurst [18, 19] attempts to select good features for a given task given a small set of evaluated features by training a linear SVM on each task using a small set of ‘base features’ as inputs, and then computing

the similarity between tasks according to the SVM weights assigned to the base features. Lee et al. [15] use additional information about features in order to learn an association between those ‘meta-features’ and the usefulness of the underlying features on tasks.

Other approaches consider feature selection in which a shared sparse set of features must be chosen for all tasks [2, 20], which is similar to SVM learning with sparsity constraints [9]. Other work in multi-task learning uses neural network architectures to structurally force all the tasks to share intermediate representations [1, 6]. An approach by Rückert and Kramer [21] on kernel learning shares our approach’s attempt to predict useful models for a target task, but concentrates on predicting a good kernel for a task using a “meta-kernel” of heuristics to compare how similar datasets are.

Compared to these techniques, our method does not assume any privileged set of ‘base’ or ‘pivot’ features, and does not require any ‘meta’ features. Our method is completely agnostic to the type of model, and does not assume kernel learning or any other specific learning framework. The core contribution of our work is to explicitly reformulate the model selection problem as collaborative filtering, which allows for a wide variety of techniques to be used unmodified, and suggests a number of interesting directions for exploration.

We initially explore the collaborative filtering approaches used by the BellKor team to (as a combined effort with two other groups) win the Netflix prize [3, 12]. As the final result was a blend of a large number of different approaches, spanning the gamut from neighborhood methods to factorization to regression techniques, their method serves as a fairly comprehensive overview of collaborative filtering techniques. For this paper we concentrate on basic factorization techniques [8, 13]. Although we consider offline factorization, where the entire ratings matrix is available at once, a strength of our method is that by taking advantage of recent work by Mairal et al. [16] on online matrix factorization and sparse coding, our method can easily extend to the online case, where the matrix is very large and continually growing through the addition of new tasks and features.

4. Method

Our method is conceptually straightforward: we start with a database of ratings, termed the *ratings store*. This database can be seen as a matrix where rows correspond to models, and columns correspond to tasks. For this paper we assume that in the store every model has been rated on every task (the matrix is complete). Then, given a new target task and a subset of rated models on that task, we use collaborative filtering techniques to predict the ratings of all the models, and return those with the highest predicted ratings as the

recommended models. We evaluate on several datasets, examples of which can be seen in Fig. 3.



Figure 3: Sample frames from the datasets, including the semi-synthetic videos that are rendered from motion capture data.

4.1. Low-level input

For this paper we consider models based on two low-level inputs: a typical STIP+HOG3D combination [11, 14], and a gridded histogram of optical flow (HOF). We use these relatively low level representation because HOF and the related HOG are widely used in computer vision and have been successfully applied to many applications.

The HOF representation divides the optical flow into $10 \times 10 \times 5$ pixel spatio-temporal cells, and a nine-dimensional HOF descriptor is computed for each cell in the standard way. So, for example, a $320 \times 240 \times 100$ video would be represented by a grid of $32 \times 24 \times 20 \times 9$ cells, and a typical trained model might be applied to a $12 \times 12 \times 10 \times 9$ scanning windows of the full grid. We use the FlowLib [7] GPU-accelerated optical flow library.

4.2. Ratings Store Generation

We generate ratings stores from both real and synthetic data. A synthetic ratings store is used to evaluate on the synthetic data, and on the real Mind's Eye data. The UCF-YT ratings store is generated from withheld UCF-YT data.

Models are generated by training SVM classifiers on the low-level input, where each classifier is trained as a 1-vs.-N classifier of actions from the dataset (synthetic or real). Tasks are generated as 1-vs.-all tasks, where each task is a binary classification problem between videos of one action vs. videos of all other actions.

Every model is rated on every task in the database using its accuracy on that task. The synthetic store uses approximately 50,000 videos to generate 10,000 models and 1000 tasks. The withheld UCF-YT data is used to produce a store of 1000 models \times 1000 tasks.

The synthetic data is used to evaluate on large libraries (10,000 models vs. 1000 for the real UCF-YT data). For this purpose, synthetic videos are rendered from the CMU motion capture database [5]. Although such synthesis cannot yet produce photorealistic data, it has seen success when used for both depth [23] and motion [17] features. Con-

sequently, we concentrate on motion and generate semi-synthetic data using our earlier work [17].

4.3. Collaborative Filtering

Our goal is to predict which trained action classifiers are likely to perform well (have high ratings) on a new unknown action, based only on the accuracies (ratings) of a small subset of those classifiers (the *probe set ratings*). We now describe how collaborative filtering techniques are used to predict the ratings of the entire library based on the probe set ratings and the ratings store. The probes are chosen at random to avoid the worst case scenario of overfitting to the training data and producing a highly redundant set.

The collaborative filtering has two parts. First, we estimate a baseline, which is intuitively the mean model ratings (average accuracy for each classifier across all actions) and mean task ratings (average accuracy of all classifiers on an action). Then, the deviations from these means (*i.e.*, *residuals*) are represented and subsequently predicted using techniques like factorization and sparse coding. The predicted rating of a model on a task (predicted accuracy of a classifier on an action) is the sum of the respective model and task means from the baseline, and the predicted residual.

Baseline Estimation

We start with a simple additive representation suggested by Koren [13], in which a model's rating on a task is represented as the sum of a global mean rating, a model factor, and a task factor. This formulation aims to capture the fact that some models are better overall than others, while some tasks are easier or harder than average. In practice, this amounts to subtracting the row and column means from the matrix. The resulting matrix of residuals is then fed into more sophisticated collaborative filtering techniques.

Formally, a rating $r_{i,j} = \mu + \phi_i + \psi_j$, where μ is a global mean rating, ϕ_i is a model-specific factor, and ψ_j is a task-specific factor. Let m be the number of models, and n be the number of tasks, so that the number of ratings is $m \cdot n$.

We estimate these factors using Koren's method [13]:

1. Estimate global mean: $\mu = \frac{\sum_i \sum_j r_{i,j}}{mn}$;
2. Initial factors: $\phi_i = \frac{\sum_j (r_{i,j} - \mu)}{n}$, $\psi_j = \frac{\sum_i (r_{i,j} - \mu)}{m}$;
3. Model factors: $\phi_i = \frac{\sum_j (r_{i,j} - \mu - \psi_j)}{n}$;
4. Task factors: $\psi_j = \frac{\sum_i (r_{i,j} - \mu - \phi_i)}{m}$.

For a new target task, we hold the previously computed model factors fixed, and estimate only the target task's factor, according to

$$\psi_t = \frac{\sum_{i \in P} (r_{i,t} - \mu - \phi_i)}{|P|}, \quad (2)$$

where ψ_t is the target task's factor, P is the set of probe features, $r_{i,t}$ is the evaluated rating of feature i on the target

task, and $|P|$ is the number of probe features.

This technique will not exactly fit the data; that is, in general $|r_{i,j} - \mu - \phi_i - \psi_j| > 0$, and the rating predicted by the simple additive method (the baseline) will differ from the observed rating. This difference, called the residual, is what the following techniques attempt to explain.

We define the residuals \bar{r} that remain after the baseline estimation by $\bar{r}_{i,j} = r_{i,j} - (\mu + \phi_i + \psi_j)$. We let \bar{R} denote the entire $(m \times n)$ residuals matrix for the source tasks. Similarly, the residuals for the target task are given by $\bar{r}_{i,t} = r_{i,t} - (\mu + \phi_i + \psi_t)$.

Factorization methods

The goal of factorization methods is to represent the residual rating of a model on a task as the dot product between a model factors vector and a task factors vector, where the dimensionality of these factors vectors corresponds to a chosen number of k latent factors. Formally,

$$\bar{R} = F^T D, \quad (3)$$

where F^T is a $(m \times k)$ matrix of model factors, and D is a $(k \times n)$ matrix of task factors. While there are many factorization schemes, a simple and popular choice is to use the singular value decomposition, in which $\bar{R} = USV^T$. Then, supposing that k latent factors are sought, S_k is the $k \times k$ upper left sub-matrix of S , and U_k is the first k columns of U . We construct the model factors matrix as $F^T = U_k S_k$, and the task factors as $D = V_k$.

Now, given a target task’s residual ratings of p probe models (without loss of generality we can assume they are the first p models), denoted \bar{r}_p , we estimate the target task’s factor vector by solving the linear least-squares problem

$$(\hat{F}^T)x = \bar{r}_p \quad (4)$$

for x , where x is a $(k \times 1)$ vector of the target task’s factors, and \hat{F}^T is a $(p \times k)$ matrix of the first p rows of F^T . Finally, predict the target task’s residual ratings for all the models:

$$\bar{r}' = F^T x, \quad (5)$$

where \bar{r}' are the predicted residual ratings. The final (non-residual) predicted ratings are produced by adding the baseline factors back to the predicted residuals, so that $r'_i = \bar{r}'_i + \mu + \phi_i + \psi_t$.

Note that if the goal is to rank the models according to their predicted ratings, it is only necessary to add the model factor ϕ_i , since μ and ψ_t are constant offsets that apply equally to all models.

Sparse Coding

Another approach is to use sparse coding, which attempts to represent the column of residual probe ratings as a sparse

linear combination of columns (tasks) from the residual ratings matrix. Sparse coding optimizes the problem

$$\arg \min_{\alpha} \|\bar{r}_p - \bar{R}_p \alpha\|_2^2 + \tau \|\alpha\|_1,$$

where \bar{r}_p are the residuals of the probe ratings after baseline subtraction, \bar{R}_p are the rows of the residuals rating matrix corresponding to the probe models, and α is the vector of weights for the sparse reconstruction, one per task in the ratings store. The parameter τ controls sparsity, with higher values of τ corresponding to increased sparsity. Once α has been computed, the predicted residual ratings \bar{r}' for all models can be computed simply as the weighted combination of columns of \bar{R} , or the matrix product $\bar{r}' = \bar{R} \alpha$. As with the factorization approach, the (non-residual) predicted rating of a model on the target task is just the residual plus the global mean, target task mean, and model mean.

In a collaborative filtering context this can be seen as a neighborhood method, where tasks corresponding to the non-zero α s are the neighbors of the target task, and the prediction is a weighted combination of the neighbors.

5. Evaluation

We evaluate in two ways. First, using synthetic data of just the ‘walking’ action, we validate our hypothesis presented in the ‘‘thought experiment’’ section.

Next, we consider the problem of recommending models across different actions, evaluating on both held-out synthetic tasks, and on real tasks pulled from the UCF YouTube 50 actions (UCF-YT)¹ and Mind’s Eye (ME)² The ME dataset is composed of outdoor actions, such as running, jogging and jumping, filmed from stationary cameras. For example frames, see Fig. 3.

5.1. Validating the thought experiment

We use the following synthetic example to validate the thought experiment discussed in the introduction. The goal is to recognize ‘‘walking’’ in video, and the only manipulated source of variation is the viewing angle. Each candidate model is a classifier trained to recognize walking from a model-specific viewing angle (for example, 33°). Likewise, each task is a synthetic classification problem, where the goal is to recognize walking from a task-specific range of viewing angles (for example, 120° – 155°); see Fig. 4. For the factorization approach, we would qualitatively expect the first two factors to encode the angles of the tasks and models. Indeed, after factorization (Fig. 5), we can see that the first two factors do encode the angles for both tasks and models. Note that while the models are arranged in an unfilled circle, the tasks form a filled circle — the edge is

¹<http://server.cs.ucf.edu/~vision/data.html>

²http://www.darpa.mil/Our_Work/I20/Programs/Minds_Eye.aspx

occupied by tasks with low angular spread, while the center is tasks with high angular spread. This is because as the angular spread of a task increases, its ‘preference’ for any one model over another decreases. At the limit, that is to recognize “walk” from any angle at all ($0^\circ - 360^\circ$), there is no reason to prefer any model over another, corresponding to factors of (0,0), which rate all models equally.

As a second example (see Fig. 2) we consider a more complicated synthetic situation with classifiers tuned to different viewing positions (angle from horizontal and distance), and tasks which vary in a similar manner. Note that even in this more realistic situation, with only 12 evaluated classifiers we are able to accurately predict the performance of the whole set of 1600. In Fig. 6 we use this example to demonstrate how recommendation is able to do better than directly rating every model.

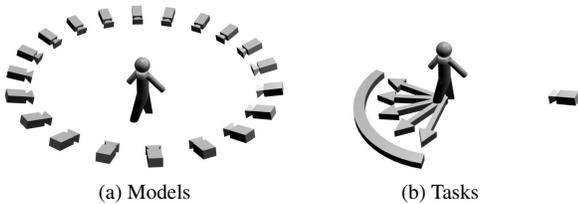


Figure 4: For one action (“walk”), models (a) are trained to recognize the action from different viewing angles. Each task (b) is likewise to recognize walking from a specific range of angles.

5.2. Cross-action recommendation

Now, we consider making recommendations for a task on an *unknown* action using a database of heterogeneous actions. So, for example, a target task might be to recognize “jump on trampoline” (although this label is not known), yet the

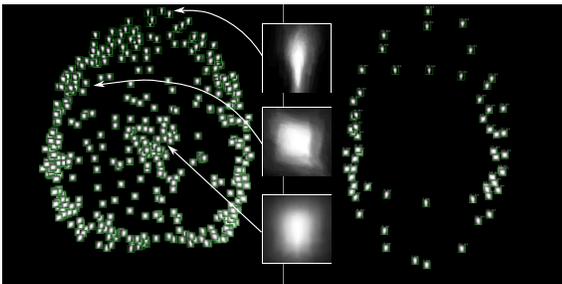


Figure 5: Scatter plot of tasks (left) and models (right), according to their first two factors. Models are arranged in a circular pattern according to the angle they were trained from, as are tasks. However, the center of the task circle is filled by tasks with very wide angular spreads, because they equally favor all models. Each “point” is the average silhouette of the positive videos for that task (best viewed under magnification in digital copy).

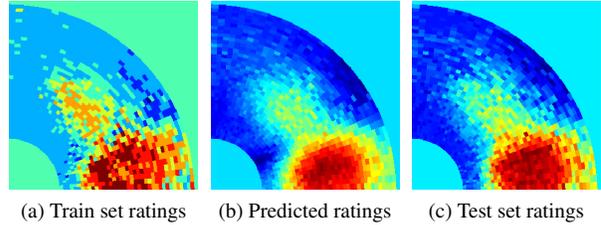


Figure 6: Recommendation improves results even when all models are rated because the model ratings on the *training* set are noisy and subject to quantization artifacts (a), especially when there are few training samples. Recommendation (b) uses these noisy ratings to produce a better prediction of the ratings on the *test* set (c). The models are arranged as in Fig. 2.

database does not necessarily contain this action, but might contain *similar* actions such as “jumping jack” or “jump for joy”. We caution that these textual labels are for illustrative purposes, and such meta-data are *not* available.

For the synthetic data we generate a 10,000 model by 1000 task ratings store using synthetic data in the standard way, with the motion capture clip used to generate each synthetic task drawn at random from the whole motion capture database, and no constraints within a task on the range of viewing angles. The candidate pool of models is generated by training classifiers on random pairs of synthesized actions. For the UCF-YT data, we use two-thirds of the dataset to generate both a library of 1000 models and a ratings store of those 1000 models evaluated on 1000 tasks.

Some factors for the synthetic data ratings store are visualized in Fig. 7. While some factor dimensions correspond to intuitive concepts (like horizontal motion), others are not so compactly described. The most obvious distinction between tasks in this factor space is between walking-type actions, and in-place actions.

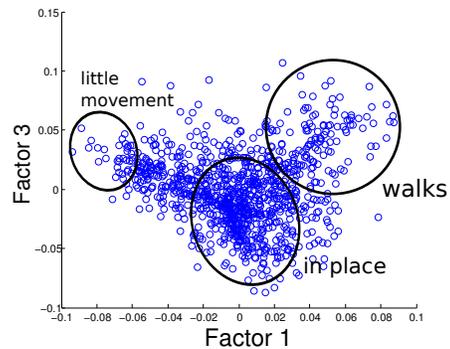


Figure 7: Visualized factors for tasks under the cross-action recommendation application.

First, we evaluate on the same type of synthetic tasks used to populate the matrix, using tasks that were held out from the matrix generation and factorization process. We use 180 evaluation tasks, where each is divided into a training portion of 8–32 samples, and a larger testing portion.

For real data, we first consider test tasks generated from the Mind’s Eye dataset, where the test tasks are created as 1-vs.-rest action classification problems, with actions drawn from the set of “walk”, “jump”, “pick_up”, and “fall_down”. So, for example, one task might be “pick_up” vs. the remainder. Each task is divided into a training set of 2–16 samples, and an evaluation set of all the remaining samples, so that the training set is highly restricted in the number of samples. Note that each ‘positive’ action appears in multiple tasks, but the tasks differ in the selection of samples. Due to the limited amount of ME data, we use the synthetic models and ratings store for the recommendation.

Another set of tasks comes from the UCF-YT dataset; since this evaluation uses all real data (two-thirds of the data is used to generate the model library and ratings store, and one-third to evaluate), we do not use the motion-only HOF features, but instead employ a typical combination of STIP [14] and HOG3D [11] in a bag-of-words formulation. We generate these tasks as 1-vs.-all classification problems, with 2–16 training samples per task.

In all cases, the objective is to pick the single best model. We compare against the natural baseline, which is to evaluate the probe set and pick the model from that set with the best performance. Since tasks vary in difficulty, to report performance across tasks we report results as mean offsets of performance vs. the mean model performance. A score of +0.0 means that the method is statistically no better than selecting a model at random from the whole pool.

Results for the ME tasks can be seen in Fig. 8. Direct selection reaches a maximum accuracy with 20 probes, and then degrades due to overfitting. In contrast, model recommendation shows an upward curve, reaching a maximum of with 180 probe models. Note that for the ME dataset at $n=180$ probe every model is in the probe set, and yet it is still better to use the recommended model rather than the model with the best direct rating (in fact, model recommendation shows the greatest advantage over direct selection when every model is chosen as a probe).

The results for the synthetic and UCF-YT tasks show similar trends, in Fig. 9a and Fig. 9b. In both cases recommendation does better with only a fraction of rated probe models than the baseline does when it rates all the models and selects the best. Interestingly, in these cases the effect of overfitting in the baseline is less pronounced, with the direct selection simply plateauing rather than noticeably decreasing. As a second baseline, for the UCF-YT data we also directly train classifiers on the STIP+HOG3D BOW histograms of the training data; this baseline obtains an accu-

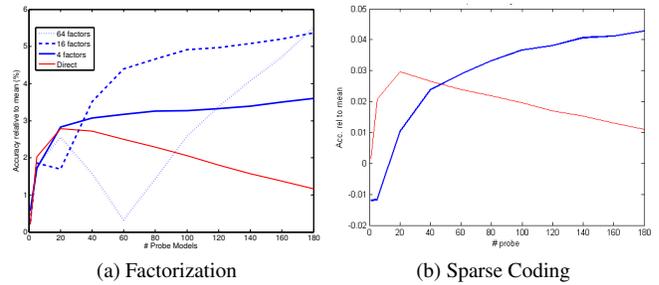


Figure 8: Effect of the number of factors used for the factorization model on accuracy vs. probe set size for the ME dataset. A larger number of factors results in a higher asymptotic accuracy, at the cost of lower performance when few probe models are evaluated. Sparse coding exhibits the same effect, but with a more graceful degradation. ‘Direct’ is the direct selection baseline.

racy of 77%, better than direct selection from the model library, but worse than model recommendation’s 78%. Since both direct training and model recommendation produce an estimate of how good their models are (direct training by cross-validation accuracy on the training set, recommendation by the predicted accuracy of the top model), we can easily combine the two by selecting the technique that reports the highest estimated accuracy on each task; this combination produces a mean accuracy of 81%.

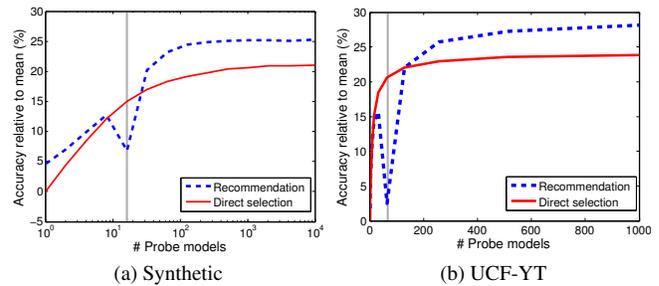


Figure 9: Mean relative accuracy vs. number of probe models for synthetic (a) and UCF-YT (b) datasets. The same trend is observed in both datasets, although the magnitude of the effect is larger in the synthetic data. At the gray line the number of probes is equal to the number of factors (16 and 64 respectively); for fewer probes, the factor estimation is underconstrained.

An interesting effect (Fig. 8) for the ME data and Fig. 10 for the synthetic tasks, is the tradeoff between the number of factors used in the factorization method and accuracy, which manifests itself as a ‘cusp’ when the number of probe models is equal to the number of factors used (16 in those cases). As the factorization method solves for the unknown factors as a linear problem, if fewer probe models are eval-

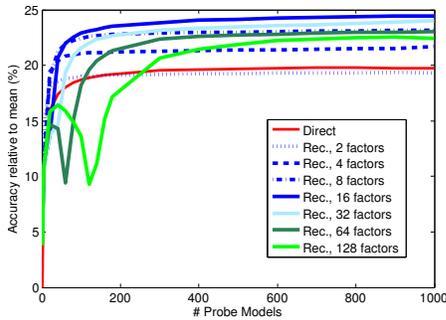


Figure 10: Effect of the number of factors in the factorization model for the synthetic tasks.

uated than factors, the problem is underconstrained, and the accuracy suffers. Hence, if 16 factors are used, then the factorization will not reach peak accuracy until 16 probe models have been evaluated. While sparse coding shows the same tradeoff (with the caveat that there is no simple relationship between τ and minimum number probe models), it does not feature this dramatic cusp (Fig. 8b).

6. Conclusions and Future Work

We exploit the intuition that the problem of selecting a good model out of a large set is analogous to the problem of recommending a consumer item and that the model recommendation problem can likewise be approached through the collaborative filtering methods used in recommendation systems. Our results confirm our counter-intuitive hypothesis that recommendation systems can select better classifiers than directly evaluating every model on the new training set.

While in this paper we have assumed that the model ratings matrix is complete, collaborative filtering techniques have been developed to deal with the highly sparse patterns of ratings assigned by human users. This prompts the question of whether it is better to recommend from a smaller but denser matrix, or a larger and sparser one, if the total number of ratings is constant. Another future direction is to extend the system to jointly recommend sets of models, rather than independently recommending individual ones.

Acknowledgments

This work was partially funded by the Army Research Laboratory under Cooperative Agreement #W911NF-10-2-0061. The views and conclusions are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for government purposes notwithstanding any copyright notation herein. We thank M. Tappen for his valuable feedback.

References

- [1] A. Ahmed, K. Yu, W. Xu, Y. Gong, and E. Xing. Training hierarchical feed-forward visual recognition models using transfer learning from pseudo-tasks. In *ECCV*, 2008.
- [2] A. Argyriou, T. Evgeniou, and M. Pontil. Multi-task feature learning. In *NIPS*, 2007.
- [3] J. Bennett and S. Lanning. The Netflix prize. In *KDD Cup and Workshop*, 2007.
- [4] J. Blitzer, R. McDonald, and F. Pereira. Domain adaptation with structural correspondence learning. In *EMNLP*, 2006.
- [5] Carnegie Mellon University Graphics Lab. CMU graphics lab motion capture database, 2001.
- [6] R. Caruana. Multitask learning. In *Machine Learning*, 1997.
- [7] A. Chambolle and T. Pock. A first-order primal-dual algorithm for convex problems with applications to imaging. Preprint.
- [8] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407, 1990.
- [9] T. Evgeniou, C. A. Micchelli, and M. Pontil. Learning multiple tasks with kernel methods. *JMLR*, 6:615–637, 2005.
- [10] R. Gopalan, R. Li, and R. Chellappa. Domain adaptation for object recognition: An unsupervised approach. In *ICCV*, 2011.
- [11] A. Kläser, M. Marszałek, and C. Schmid. A spatio-temporal descriptor based on 3D-gradients. In *BMVC*, 2008.
- [12] Y. Koren. The BellKor solution to the Netflix Grand Prize. http://www.netflixprize.com/assets/GrandPrize2009_BPC_BellKor.pdf, 2009.
- [13] Y. Koren. Factor in the neighbors: Scalable and accurate collaborative filtering. *ACM Trans. KDD*, 4(1):1:1–1:24, 2010.
- [14] I. Laptev, M. Marszałek, C. Schmid, and B. Rozenfeld. Learning realistic human actions from movies. In *CVPR*, 2008.
- [15] S.-I. Lee, V. Chatalbashev, D. Vickrey, and D. Koller. Learning a meta-level prior for feature relevance from multiple related tasks. In *ICML*, 2007.
- [16] J. Mairal, F. Bach, J. Ponce, and G. Sapiro. Online learning for matrix factorization and sparse coding. *JMLR*, 11, 2010.
- [17] P. Matickainen, R. Sukthankar, and M. Hebert. Feature seeding for action recognition. In *ICCV*, 2011.
- [18] I. Mierswa and M. Wurst. Efficient case based feature construction for heterogeneous learning tasks. In *ECML*, 2005.
- [19] I. Mierswa and M. Wurst. Efficient feature construction by meta learning – guiding the search in meta hypothesis space. In *ICML Workshop on Meta Learning*, 2005.
- [20] G. Obozinski and B. Taskar. Multi-task feature selection. In *ICML Workshop on Structural Knowledge Transfer for Machine Learning*, 2006.
- [21] U. Rückert and S. Kramer. Kernel-based inductive transfer. In *ECML*, 2008.
- [22] K. Saenko, B. Kulis, M. Fritz, and T. Darrell. Adapting visual category models to new domains. In *ECCV*, 2010.
- [23] J. Shotton, A. Fitzgibbon, M. Cook, and A. Blake. Real-time human pose recognition in parts from single depth images. In *CVPR*, 2011.