Model Recommendation for Action Recognition and Other Applications

Pyry Matikainen

CMU-RI-TR-12-36

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Robotics

The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

December 2012

Thesis Committee:
Martial Hebert, Co-chair
Rahul Sukthankar, Co-chair
Yaser Sheikh
Ivan Laptev, *INRIA*

Copyright © 2012 by Pyry Matikainen. All rights reserved.

Keywords: action recognition, collaborative filtering, recommender systems

For Sini

Abstract

The typical approach to learning based vision has been that for each individual application, classifiers or detectors are learned anew from annotated training data for each specific task. However, the classifiers trained in this way tend to be brittle and highly specialized to the datasets from which they are derived, making them difficult to transfer between tasks. While multi-task learning and domain adaption techniques address some of these problems on a theoretical level, from a practical standpoint they are just as complicated and labor-intensive as the simpler learning techniques they supplant.

However, suppose that these specialized classifiers had simply been *collected* into a library: while it is unlikely that any *specific* classifier would generalize well to a new dataset, there may exist some classifier in the library that is tuned to the same conditions as the new task. This thesis addresses the fundamental question of how to efficiently select a good classifier from such a library.

Specifically, this thesis demonstrates that collaborative filtering techniques (such as employed by recommender systems like Netflix and Amazon.com) can be used to *recommend* models appropriate for a specific target task. These recommendations are made by trying, or rating, a small subset of models on the target task, and then using that small set of ratings along with the ratings of the models on other tasks to predict the ratings of the unevaluated models on the target task.

This process, which we term "model recommendation", is applied to action recognition and other vision and robotics applications, and the subtle differences between model recommendation and typical recommender systems are used to derive novel algorithms and extensions to the core recommendation concept.

Acknowledgments

First, I would like to thank my advisors, Martial Hebert and Rahul Sukthankar for putting up with me for almost six years, and especially Rahul, who's shown admirable dedication to meeting with me every week despite traveling between at least three different locations and moving to Google. Then everyone in the dinner train (Pras, Brian, Mike, Heather, Joydeep, Nate). Scott Satkin and Kris Kitani for getting me last minute figures and data. Finally, Saint Steven for giving me hope.

Contents

1 Introduction **1**
- 1.1 The Parallel Between Recommender Systems and Model Libraries 4
- 1.2 General Terminology and Notation 5
  - 1.2.1 Classification Tasks 6
  - 1.2.2 Models and Model Ratings 6
- 1.3 Differences Between Model Recommendation and Consumer Product Recommendation 7
- 1.4 Contributions 8

2 Literature Review **11**
- 2.1 Choosing Features: Feature Selection and Dictionary Learning 11
- 2.2 Sharing Features and Training: Domain Adaptation and Transfer Learning 12
- 2.3 Sharing Intermediate Representations: Multi-Task Learning 13
- 2.4 Sharing Models: Selection from a Library 14
- 2.5 Collaborative Filtering 15

3 Applications and Datasets **19**
- 3.1 Action Recognition 19
  - 3.1.1 Common Low-level Features 19
  - 3.1.2 UCF50 and UCF11 20
  - 3.1.3 UCF50 (ActionBank features) 21
  - 3.1.4 Mind's Eye 21
  - 3.1.5 Semi-Synthetic Rendered Motion-Capture 22
- 3.2 3D Scene Model Matching 25
  - 3.2.1 Models 25
  - 3.2.2 Tasks 25
  - 3.2.3 Ratings 25
- 3.3 Skin Detection in Egocentric Video 26
  - 3.3.1 Models 27
  - 3.3.2 Tasks 27
  - 3.3.3 Ratings 27
- 3.4 Robot Controller State-Machines 28
  - 3.4.1 Simulator and State-Machine Controllers 28
  - 3.4.2 Tasks 30

3.4.3 Models . . . 30
3.4.4 Ratings Store . . . 31

4 Action Recognition **33**
4.1 Base Descriptor: Trajectons . . . 34
4.2 Alternate Base Descriptor: STIP-HOG . . . 35
4.3 Augmenting Descriptors with Pairwise Relationships . . . 36
4.3.1 Pairwise Discrimination with Relative Location Probabilities (RLPs) . . . 36
4.3.2 Estimating Relative Location Probabilities from Training Data . . . 38
4.3.3 Extension to Temporal Relationships . . . 39
4.3.4 Classification . . . 40
4.4 Evaluation . . . 40
4.4.1 Effect of Temporal Relationships . . . 42
4.4.2 RLPT Sparsity . . . 42
4.5 Conclusions . . . 43

5 Feature Seeding **45**
5.1 Overview . . . 47
5.2 Feature Pool Generation and Evaluation . . . 47
5.3 Feature Seeding/Filtering . . . 49
5.4 Evaluation . . . 49
5.4.1 Feature Statistics . . . 50
5.4.2 Comparison with Other Quantization Methods . . . 51
5.4.3 Comparison of Base Descriptors . . . 52
5.5 Conclusion . . . 54

6 Single Model Recommendation **55**
6.1 Estimated vs. Ideal ratings . . . 55
6.2 Collaborative Filtering Algorithms . . . 56
6.3 Interpretations of Factorization-Based Recommendation . . . 59
6.3.1 Factorization as Finding Latent Factors . . . 59
6.3.2 Factorization as Projection Onto a Basis . . . 61
6.4 Quantitative Results . . . 62
6.4.1 Mind's Eye Tasks . . . 63
6.4.2 UCF50 and Semi-Synthetic Motion-Capture Tasks . . . 63
6.5 Search vs. Smoothing . . . 67
6.5.1 Unified Search-Smoothing Algorithm . . . 69
6.5.2 Quantitative Results . . . 69
6.5.3 Effect of the α Parameter . . . 72
6.6 Regularized Least Squares and the "Cusp" . . . 74

7 Ensemble Recommendation **77**
7.1 Ensemble Recommendation Methods . . . 78
7.1.1 Top-k Recommendation . . . 78
7.1.2 Set Expansion . . . 78
7.1.3 AdaBoost . . . 79
7.1.4 Recommendation Boosting . . . 79
7.1.5 Recommendation Boosting+ . . . 80
7.2 Qualitative Demonstration . . . 80
7.3 Quantitative Evaluation . . . 82
7.4 Redundancy . . . 83
7.4.1 Defining Redundancy . . . 83
7.4.2 Artificially Increasing Redundancy . . . 84
7.4.3 Measuring the Redundancy of Real Applications . . . 85

8 Incomplete Ratings **87**
8.1 Recommendation from Incomplete Ratings . . . 87
8.1.1 Factorization with Incomplete Ratings . . . 87
8.1.2 Evaluating the Cost of Incompleteness . . . 88
8.1.3 Store Size vs. Completeness . . . 88
8.2 Probe Selection . . . 90
8.2.1 Optimal Design of Experiments . . . 90
8.2.2 Relationship of Factorization Based Collaborative Filtering to Optimal Design of Experiments . . . 91
8.2.3 Evaluating Probe Selection . . . 92

9 Sequential Selection **95**
9.1 Related Work . . . 96
9.2 Neighborhood Collaborative Filtering with Variance Estimates . . . 97
9.3 Evaluation Scenarios . . . 97
9.4 Strategies . . . 98
9.4.1 Random . . . 99
9.4.2 Batch Recommendation . . . 99
9.4.3 Greedy Recommendation . . . 99
9.4.4 Upper Confidence Bound Bandit (without recommendation) . . . 101
9.4.5 Bandit Recommendation . . . 101
9.4.6 Neo-Bandit Recommendation . . . 101
9.5 Incomplete Database . . . 103
9.6 Results . . . 103
9.7 Conclusions . . . 106

10 Conclusions and Future Directions **107**
10.1 Summary of Contributions . . . 107
10.1.1 Trajectons and Feature Seeding . . . 107
10.1.2 Model Recommendation . . . 107

10.1.3 Ensemble Recommendation . . . 108
10.1.4 Sequential Selection . . . 108
10.1.5 Incomplete Ratings . . . 108
10.2 Avenues for Future Exploration . . . 108
10.2.1 Adapting Recommended Models . . . 108
10.2.2 Ratings Store Growth . . . 109
10.2.3 Real Applications . . . 109
10.3 Concluding Thoughts . . . 110

11 Publications **113**

Bibliography **115**

Chapter 1

Introduction

People unfamiliar with computer vision often seem to think that vision researchers are hording an extensive collection of detectors, which presumably sit in the same mythical "back room" that contains out-of-stock items in retail stores. They are disappointed to learn that simply because a researcher may have written a paper on (say) classifying dogs vs. cats does not mean that the researcher actually *has* a generic "cat" detector on hand. No, instead what the researcher typically has is a convoluted and error-prone method for *generating* a cat detector for a specific dataset, provided that "enough" training data is available. It is doubly disappointing that these classifiers tend to be brittle and highly dataset specific, so even if a user trains a detector that seems to work well for their particular task (properly cross-validated and everything), it still might not work well in practice.

But is this expectation really so out of line? It actually seems rather reasonable to expect that a researcher working on action recognition might have some general action detectors laying around.

Naive users may expect that just as there are competing algorithms for, say, feature tracking that simply work out of the box, that the competing algorithms for action detection should likewise work. That is, users expect that they should be able to *shop* for detectors or classifiers, and are disappointed to find that the algorithms are in fact merely frameworks for training specialized classifiers for their specific tasks.

But supposing there *were* such a library of detectors and classifiers, how would users actually go about choosing a classifier from the library? We have already accepted that the classifiers are likely to be highly dataset specific, for example, so that a "walking" detector trained from a high viewing angle is unlikely to perform well at detecting walking from a low viewing angle. In other words, even if such a library existed, it is unlikely that users could simply rely on textual labels or other such classifier meta-data in order to choose the classifier best for their specific tasks.

A more direct approach is that the user could simply *try* classifiers from the library, and then choose the classifier which performs best. This parallels the approach they might take if they were choosing a feature tracker, which is to download as many implementations as they can find, try them all, and choose the one that works best, either as determined by a subjective evaluation, or according to quantitative performance on a validation dataset.

To return to the analogy of shopping for a classifier, how does a user looking for, to use a

non-random example, a movie to watch choose which movie they are going to rent? Obviously, trying every movie would make no sense, not least because there are thousands of movies to choose from. In previous decades, users would have relied on aggregate ratings or the opinions of friends and critics, but today there is an increasingly popular option: namely, recommender systems. In a recommender system, the user provides feedback on how much they like a subset of all possible movies, and based on how the user *rated* those movies, the system produces a recommendation for what movies the user is likely to enjoy.

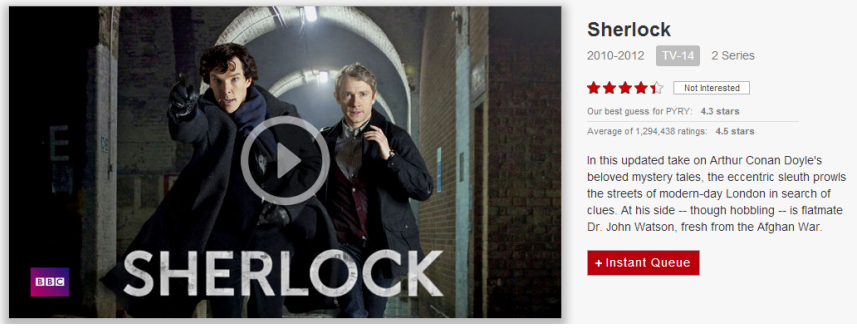


Figure 1.1: Example of Netflix.com's recommendation system: the predicted rating for a user (PYRY) is not simply the average across all the other users.

These recommendation systems are driven by collaborative filtering techniques which use all the users' ratings data to make predictions for individual users. For example, Netflix.com predicts what users will rate movies they have not seen (Fig. 1.1), and this prediction is not simply an average rating across all other users. This idea of using the collective *experience* of users (as reflected in their ratings) to personalize recommendations for individuals is a very powerful one, and there is a surprising connection between these types of consumer recommender systems and the problem of selecting good classifiers from a library tuned for a specific task: namely, that once distracting labels like "users" and "movies" and "tasks" and "classifiers" are stripped away, the problems become nearly identical.

This surprising parallel suggests that the collaborative filtering techniques employed for consumer item recommender systems could be easily re-purposed to the problem of selecting classifiers, or indeed, more general "models" from large libraries, where the selection does not pick based on some aggregate average, but instead is tuned to recommend good models *specifically* for different tasks. This thesis introduces the idea of using collaborative filtering for selection from model libraries, and coins the term "model recommendation" for the process.

Models are in some sense digital tools that operate on tasks, and the rating of a model on a task does not reflect a preference, but rather how *useful* that model is on the task. Fig. 1.2 shows some examples of what could be considered models operating on video-based tasks. The rating of each of these representations would be a measure of how useful they are for a specific task; for example, if the task is video classification, the rating of each model could be the accuracy of an SVM trained on that representation on the videos in the task.

However, beyond merely applying collaborative filtering techniques to computer vision and robotics, this thesis investigates the subtle differences between typical consumer item recommender systems and model recommendation, and shows how these differences can lead to sur-

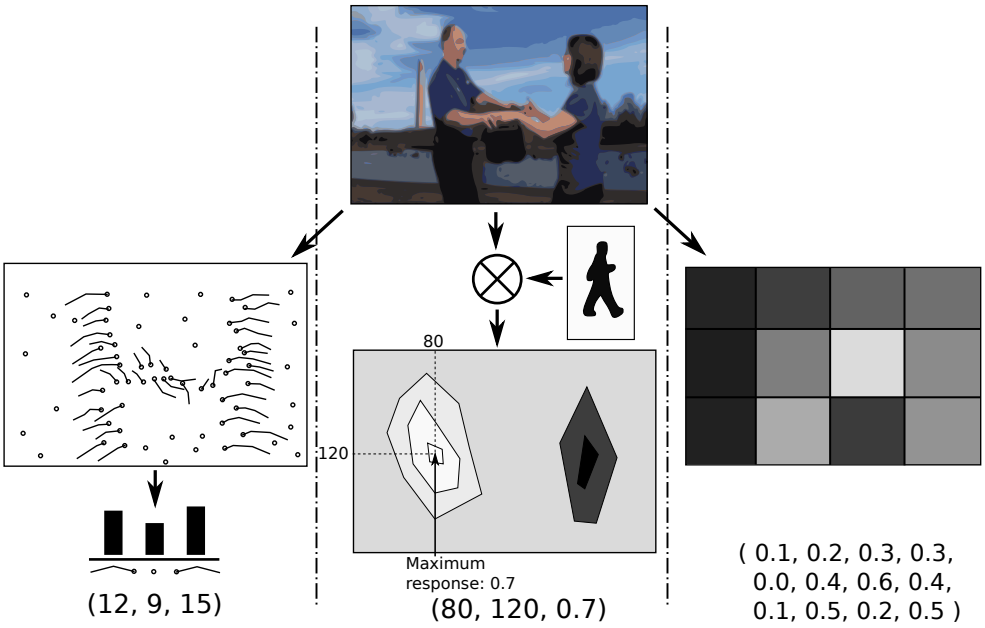


Figure 1.2: Examples of what can be considered *models* in model recommendation. On the left, a histogram of motions is computed from the video. In the center, the maximum response and location of that response for a motion template comparison is computed. On the right, the video is gridded, and the magnitude of the motion in each cell is computed.

prising conclusions (such as the result that a recommender system can recommend a better model or classifier than even if *every* model in the library is tried) and novel algorithms (such as recommendation boosting and recommendation bandits).

To say that this is a work targeted at datasets that do not yet exist is true but only half of the story. For the very word "dataset" suggests that existing monolithic approach in which training data and eventual application are kept well apart. But in collaborative filtering, in Netflix for example, which users are the training users and which are the test users? The collaborative part of collaborative filtering is that each user improves the recommendations for each other user by improving their own results: users rate movies because they want better recommendations, not out of altruism. In comparison, in a traditional dataset there is no benefit to the annotator, which is why they are either unpaid students, or paid Mechanical Turk users.

Ratings as *performance measures* are fundamentally different from typical annotations: a user has no interest in actually annotating their data, because annotating your data to produce a classifier to annotate your data is pointless! However, giving feedback on the performance of classifiers in order to improve that performance is a much less demanding type of information to produce.

Collaborative filtering grew significantly as a field as a result of the Netflix prize: the application motivated the theory. Although such datasets do not yet exist for model recommendation, computer vision is now seeing the growth in real-world applications that will motivate the collection of the *performance data* or *ratings* that drives model recommendation. In other words, the datasets that this method is built for do not exist yet because they are just now being collected.

1.1 The Parallel Between Recommender Systems and Model Libraries

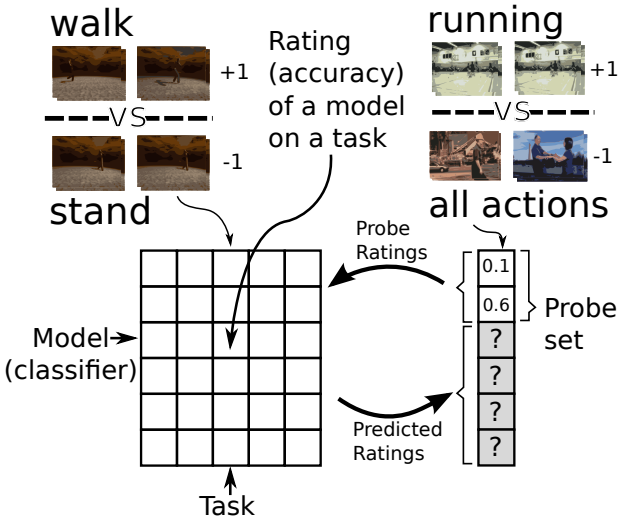


Figure 1.3: The goal of model recommendation is to predict the accuracy of models (classifiers) on a task based on how well a *probe set* of classifiers perform on that task, and a database of how well the classifiers perform on other tasks. In this way a good classifier can be selected from a large pool by only testing a small number of classifiers from the pool.

At the center of this thesis is the realization that the problem of selecting models or classifiers tuned for specific tasks is exactly analogous to the problem of recommending items for consumers. Since recommender systems have largely moved to collaborative filtering techniques that only use the *ratings* that users give items, and not any user or item specific information, these techniques are more general than they are often given credit for. The central analogy of this thesis is that the model selection problem can be seen as a recommendation problem, hence the name "model recommendation".

A standard collaborative filtering setup considers the problem of predicting the *ratings* that *users* would give to *items* in a library, if they were to rate them, and given that they have already rated some items in the library. Typically this setup is imagined as an incomplete matrix, where items correspond to rows and users to columns, and a given element r_{ij} in the matrix R of ratings corresponds to the rating a user j has assigned to an item i, and where not all entries are known. Common collaborative filtering approaches are neighborhood techniques, where the unknown ratings of a user are predicted from the "neighbors" of that user according to their known ratings, and factorization techniques, which make low-rank assumptions on the matrix R in order to fill in the missing entries. These methods are called factorization because the low rank assumption is equivalent to factorizing the matrix R into a product of two low-rank matrices which, when multiplied, explain the visible ratings well and predict the missing entries.

Fig. 1.3 illustrates a recommendation scenario: the objective is to predict the ratings which a user will give to the items in a *library*, based on how that user has rated a subset of those items. In

model recommendation, the scenario is exactly the same, except instead of a library of consumer items, there is a library of *models* (e.g., classifiers) which are rated by *tasks* instead of users.

1.2 General Terminology and Notation

This section describes in more detail the general problem setting and related terminology (see Fig. 6.4).

In the most general case, there is not a formal definition for either a *task* or a *model*, all that is required is that there be a method of obtaining a model's *rating* on a task. The *rating* is a numerical score of the performance of that model on the task, where a higher score is 'better', for whatever notion of 'better' is relevant to the application from which the tasks are drawn. The following section discusses *classification tasks* to give a concrete example of a type of task and ground the remainder of the thesis, but tasks need not be simple classification. In the sections on 3D model matching (Sec. 3.2) and robotic vacuum cleaner policy recommendation (Sec. 3.4), applications are discussed in which the tasks are not classification and the models are not classifiers. The following terms are used throughout the thesis:

- Tasks (= users): Tasks are analogous to users, they are what models are being recommended "for".
- Models (= items): Models are algorithmic objects, for lack of a more precise definition. As discussed later, there is not a formal definition for either models or tasks, since they are only considered in relation to one another (i.e., the capability of rating models on tasks)
- Ratings: A rating of a model on a task is a measure of how well that model performs on the task. The rating is a real number. The objective of model recommendation is to predict the ratings of models on a new task.
- Ratings Store: A matrix of the ratings of the models in the library on different tasks. This matrix can be incomplete, that is, it can be missing ratings. By convention, the matrix is $n \times m$, where n is the number of models in the library, and m is the number of store tasks.
- Probe Set: The set of models with known ratings on the test task.
- Test Task or Target Task: The task for which models are being recommended / having their ratings predicted.
- Store Task: A task in the ratings store; *i.e.*, a column of the ratings store.
- Model Library: The set of models from which recommendations are made.

To put this terminology in context, in model recommendation first a *ratings store* of the *ratings* of the *models* in the *model library* on the *store tasks* is built. This *ratings store* is represented as a matrix, where each entry corresponds to the *rating* of a different *model* on a *store task*. For example, if the models are classifiers, and the tasks are labeled datasets, then each rating (matrix entry) is the accuracy of a specific classifier on a specific dataset. Given this ratings store, for a new *test task* where only a subset (the *probe set*) of models have been rated on the task, the objective is to predict the ratings of all the models in the library, and thereby *recommend* the model that is likely to perform best on the test task. By the analogy, in the case of Netflix a new test user comes in having rated only a probe set of a few movies, and based on how that user has

rated the probe movies, the system predicts what that user is likely to rate the other movies in the library according to the ratings store of how other users have rated movies. Likewise, in model recommendation for a library of classifiers, a new test task might come in, and based on the accuracies that a few probe models from the library achieve on that task, the system recommends which classifiers are likely to be the most accurate for the new task.

1.2.1 Classification Tasks

A common type of task is a classification task, in which a task T_j is assumed to be a self-contained forced choice classification problem. Denote a task

$$T_j = \{(x_{j,1}, y_{j,1}), (x_{j,2}, y_{j,2}), \ldots\}, \tag{1.1}$$

where $x_{j,z}$ is the zth 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, complicated representation (*e.g.*, a video clip), while the target label $y_{j,z}$ is a discrete value indicating which class a given data sample belongs to. Note that labels are not consistent across tasks; there is no necessary correlation between the labels in any tasks. Different tasks can have different numbers of class labels. Also note that the data samples associated with a task need not be unique; the same data sample might be shared across many tasks (for instance, if one dataset is used for many different tasks). For simplicity, the remainder of this proposal assumes that target labels are binary; that is, that $y_{j,z} \in \{-1, 1\}$.

1.2.2 Models and Model Ratings

The term 'model' is not meant to denote any specific formal structure, but instead to mean some method or *function* that operates on tasks (however they are mutually defined). In the case of models which are classifiers, "operating on" means classifying, but there are many possible ways the models could operate (for example, in Sec. 3.4 the models are state machines which control the behavior of robots).

Here it is important to distinguish between a model's *output* and a model's *rating*. For example, a classifier model's immediate output will be a decision value, but its rating will be an accuracy. That is to say, the rating of a model is a measure of its performance over an entire task, not its direct output. As another example, if a model is a state machine which controls a robot, then the model's immediate output might be instructions to the robot's motors, while its rating on a particular task will be how well the robot performed in that task.

The rating of a model on a task is denoted by $R(f_i, T_j) = r_{ij}$. A higher rating is meant to convey that a model is in some sense better on a task. If $r_{ij} > r_{kj}$, then model f_i should give better performance on task T_j than model f_k.

Even in the case of classifier models, this rating function might be implemented in many ways. One approach would be to use one of many information-theoretic measures, such as mutual information or the "relief" measure [58], to gauge how much information the classifier's decision values give about the data sample labels. However, preliminary work with simple features suggested that the straightforward classification accuracy is a more useful rating for classifiers than mutual information.

1.3 Differences Between Model Recommendation and Consumer Product Recommendation

Although the analogy between model recommendation and traditional recommendation systems is exact in a mechanical sense, in that the same basic algorithms can be applied to both situations, there are a number of key differences between the scenarios, and a key contribution of this thesis is to explore those differences and how they may be used to the advantage of model recommendation.

Typical recommender systems are largely passive: they are simply given the items, users, and ratings, and can only passively predict the missing ratings. In contrast, a model recommendation system can take an active role, since it can control the generation of models as well as which ratings are evaluated.

Perhaps the chief difference between the two is that in a consumer recommendation system the ratings provided by users cannot be contested; if the system predicts a rating for an item that the user then actually rates a different way, then the system is in error for all practical purposes. While in a philosophical sense it is possible that a user could be mistaken about their own enjoyment of (*e.g.*,) a movie, from a practical standpoint, a user is unlikely to appreciate a system that contests their stated preferences.

The same is not true, however, of the ratings given to models on different computer vision and robotics tasks, and the reason is that the tasks against which the models are actually rated are in some sense mere *proxies* for the hidden 'true' tasks of interest. For example, suppose that the application is hand detection: the true task might be to detect human hands in all possible poses under task-specific illumination conditions. However, it is impossible to acquire a dataset of literally every hand pose configuration in order to measure the true performance or rating of a hand detector. Instead, the hand detector can be rated against a *training set*, a necessarily extremely sparse sampling of the true task of interest, and a sampling which in all likelihood is severely biased by the methodology by which the samples were obtained.

Another way of describing this problem is as overfitting, where a classifier is optimized to do well on a training set, but that performance does not transfer over to the test set or real task of interest. In contrast, a human being cannot 'overfit' their stated ratings of movies, because they (philosophical grumblings aside) can directly query their enjoyment of a particular movie rather than having to resort to measuring their enjoyment of a proxy.

In practice, this means that for a consumer recommendation system, the recommender can only ever hope to do to as well as if the person had directly rated every item themselves, but in a model recommendation problem, it is possible (in both theory and practice, as we will show) for the recommender to produce a *better* recommendation (as measured on the hidden test set) than by selecting the model with the best measured performance on the training set.

Another difference is that in a consumer recommendation problem, the system must passively accept whatever ratings the user provides, other than perhaps giving some slight incentives for users to rate more items. But the key point is that a consumer item recommendation system has no effective way to force users to rate specific items. In model recommendation, the system can force models to be rated on tasks, because there are no fickle humans in the loop– the only cost is computation time. This has a number of effects, foremost of which is that in model

recommendation it is possible to obtain a dense matrix of ratings, where every model has been rated on every task, whereas for consumer recommendation systems, very sparse ratings matrices are the norm.

Furthermore, even in the case where it is computationally impractical to rate every model on every task, a model recommendation system can still choose *which* ratings to compute, in contrast to a consumer recommendation system where the system has little control over which sparse ratings a user chooses to provide. This means that in a model recommendation problem, it is possible to choose which ratings to compute in order to produce the best predictions (see Chapter 8) for a discussion on how this selection can be performed in a batch recommendation context, and Chapter 9 for the extension of the model recommendation problem to making sequential recommendations).

A related point is that in model recommendation, the knowledge of which models have been rated does not provide any additional information, unlike in consumer recommendation systems where these "implicit ratings" by themselves can reveal a great deal about consumer preferences. For example, in the case of Netflix, merely knowing which movies a user has rated provides information about that user's preferences, because users do not watch movies at random, but tend to watch movies that they expect they will enjoy. The same is not true for a model recommendation system, since the system does not know anything about models prior to actually running them.

Typical recommender systems often prefer ratings that are *explainable*, that is, that the system can produce a description that will allow a human user to understand why the system made the recommendations that it did. For example, a collaborative filtering approach based on item-item neighbors can say that it recommended a certain item because the user rated specific other items highly. In comparison, a system based on user-user neighborhoods has more difficulty explaining its ratings, because the answer is that the user rates items in a similar way to another, anonymous user. Factorization methods have the most trouble, since usually the computed item and user factors have no obvious semantic meaning. In model recommendation there is little benefit to being able to explain the ratings, so there is no restriction on techniques on that front.

1.4 Contributions

The core contributions of this work are to make the relationship between collaborative filtering recommender systems and computer vision and robotics tasks explicit, and to develop novel extensions to the recommendation concept in order to meet the unique challenges of different vision and robotics applications.

- Selecting good features for action recognition from synthetic data: We demonstrate how semi-synthetic data (rendered motion capture) can be used to select good features for action recognition (Chapter 5).
- Recommending action classifiers from highly impoverished training data: We demonstrate that in an action recognition problem where there is very little training data (2–16 training samples), the measured accuracies of classifiers are very noisy, yet model recommendation is able to smooth those noisy ratings to produce recommendations that are *better than trying every model*, as demonstrated in Chapter 6.

- Jointly recommending sets of classifiers: We present a recommendation boosting algorithm (Chapter 7) for *jointly* recommending sets of classifiers, and show that it is resilient against even significant redundancy in the classifier library.
- Adapting recommendation to efficiently search 3D models: We show that there is a continuum between applications where the ratings are especially noisy (action classification with little training data), and applications where the ratings are known very well (matching 3D models to images). We present a model recommendation algorithm which can be tuned to accommodate both ends of the continuum, to either "smooth" noisy ratings or efficiently "search" accurate ones, and demonstrate it on a 3D model matching problem (Chapter 6.5).
- Providing practical suggestions on building ratings stores: We show that there is little benefit to using a larger but less complete ratings matrix vs. a smaller but complete one, provided the number of ratings is constant (Chapter 8).
- Making sequential recommendations for a robot's state machine controller with an exploration-exploitation tradeoff: We present a novel algorithm to recommend movement policies (implemented as state machines) for a robot in a floor coverage problem. This algorithm is presented in Chapter 9, where the problem is formulated a multi-armed bandit problem in order to address the exploration-exploitation tradeoff inherent in the scenario, and our "recommendation bandit" incorporates aspects of both collaborative filtering and multi-armed bandit solutions.

Chapter 2

Literature Review

In a broad sense this thesis is about avoiding the typical learning problem, and so this chapter starts by reviewing other ways to sidestep standard learning, and then discusses the collaborative filtering methods which drive recommender systems in more detail.

Avoiding traditional learning requires obtaining supplemental information, or in a way *sharing*, from another source, whether this information takes the form of priors, features, training samples, or entire models in our case. Thus, we have organized the related work along the lines of what is shared.

2.1 Choosing Features: Feature Selection and Dictionary Learning

Guyon and Elissee [39] present a taxonomy of feature selection, in which methods can broadly be divided into "wrapper" and "filter" methods. In a wrapper method, the feature selection incorporates the end learning, wrapping around it; for example, by greedily selecting features in turn by which ones increase the accuracy of an SVM trained on them. In contrast, in a filter method, features are selected without knowing the target classifier, for example, by looking at the correlation coefficients or mutual information between individual features. It may seem that wrapper methods are clearly superior, but in practice filter methods are frequently preferred due to their better robustness against overfitting than wrapper methods [39, 60, 105].

In our chapter on feature seeding (Chapter 5) we use a feature ranking technique [39, 105] that is inspired by boosting-based methods [27, 103, 105]. However, since we do not assume that the specific task is known on the target data, we do not rank features by their performance on a single task, but instead on aggregate performance over a basket of independent randomly generated tasks.

Conceptually related to feature selection is supervised dictionary learning. In practice, supervised or class-aware methods dictionary learning methods have shown highly variable improvements, ranging from substantial [11, 65, 110] to modest [10, 57]. As Boureau *et al*. [10] demonstrate, the overall performance of a system can be affected by the individual components in complex ways, accounting for much of this variability. That is to say, because dictionary learning methods share at such a low level, it can be difficult to empirically measure their benefits

However, it seems that in general, techniques that rely more heavily on individual features are more likely to benefit from class-aware dictionaries. For example, Brendel *et al*. [11] associate a single feature with each frame in a video, and they find that producing a separate dictionary for each action gives a substantial improvement over a shared dictionary. However, the difficulty is that a dictionary tuned to one class may actually degrade performance on others [97]. Of course, this is to be expected: metaphorically, a general English language dictionary is not likely to cover technical quantum physics terms well, and vice-versa. The specialization which improves performance on one class must come at the cost of not representing other classes as well.

In feature seeding (Chapter 5), we select our "dictionary" from a pool of features. These features are computed from raw descriptors, which may take many forms; we consider trajectory fragments [68] and space-time interest points [53]. The selection process can be seen as a filter method, where the features are chosen which perform best across a wide range of synthetic (rendered motion capture) tasks. This is similar to the approach Pinto *et al*.take in still image analysis, where synthetic data is used to evaluate the performance of algorithms [80].

2.2 Sharing Features and Training: Domain Adaptation and Transfer Learning

Another broad category of learning methods attempt to share not features, but training data across many tasks in order to get the most out of limited data.

Domain adaptation techniques attempt to adapt data from one (typically related) domain to another. These methods can be powerful across limited and well-characterized domains, such as in [50]. However, the gains are often modest, and as the aptly titled work "frustratingly simple domain adaptation" by Daumé [29] shows, even simple techniques can outperform sophisticated domain adaptation methods. Likewise, transfer learning methods such as transductive SVMs [26] can provide modest benefits, but are often computationally expensive and often restricted to datasets with shared classes. In particular, transductive SVMs, which use unlabeled data in addition to labeled data, are highly non-convex and subject to strong local minima. This means that the quality of the labeled data for transductive SVMs is particularly critical, and they do not degrade well to very few labeled training samples or noisy labels.

Typical domain adaptation techniques assume that the specific task is the same between the source and target domains [6, 29, 50], and this assumption common in transfer learning as well [26, 27, 32]. That is to say, if the problem were action recognition, then these techniques would need the specific actions of interest to be explicitly matched across the domains. For example, Cao *et al*.perform cross-dataset action detection, using one dataset (KTH) [92] to improve performance on a related one (MSR Actions Dataset II) [14]. However, the particular actions that are detected are present in both datasets, and indeed MSR was actually constructed to share those actions with KTH.

Likewise, Lai and Fox [50] use the Google Sketchup database of 3D models to match laser scans using domain adaptation, but (perhaps obviously) this technique is limited to those objects that are present in the sketchup database.

Many methods, such as those by Gopalan et al. [38] and Saenko et al. [87], use a source

domain to improve performance in a target domain by explicitly mapping samples from the source to the target. In the case where the tasks are related through a known semantic hierarchy, Fergus *et al*. [35] use the known semantic similarity between tasks according to their location in the WordNet hierarchy to transfer samples *weighted* by that similarity between tasks (classes). However, in the model recommendation problem the training samples for the models are not available, excluding this type of transfer in the general case. Furthermore, unlike the Fergus *et al*.work, model recommendation does not rely on an external guide (like WordNet) to the similarity between tasks.

In the area of "meta-learning" a number of approaches are conceptually related to our model recommendation. Blitzer *et al*.use a set of 'pivot features' [9] 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, an approach by Mierswa and Wurst [72, 73] 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. Similarly, Lee et al. [54] use additional information about features in order to learn an association between those 'meta-features' and the usefulness of the underlying features on tasks.

These bear a mechanical similarity to neighborhood based collaborative filtering in which users or items are compared to one another by their ratings. However, a key difference is that modern collaborative filtering methods do not assume that there is any privileged set of 'pivot', 'base', or 'meta' features. Interestingly, early collaborative filtering methods *did* make this assumption, for example in Goldberg *et al*.'s Eigentaste algorithm [37].

2.3 Sharing Intermediate Representations: Multi-Task Learning

More closely related to our method are techniques in multi-task learning, where the objective is to learn (for example) classifiers on multiple tasks simultaneously, typically by forcing the learned classifiers to share some intermediate representations.

A related idea is that of learning from *pseudo-tasks*, as in Ahmed *et al*. [1], where the learning of mid-level features is regularized by penalizing features for poor performance on a set of artificially constructed pseudo-tasks. The pseudo-tasks are simple image-processing problems (*e.g*., , predict the maximum response of a filter over an image), so forcing the learning to produce mid-level features that perform well at them in some sense constrains the intermediate features to be useful for general image processing tasks. Our synthetic data (Chapter 3.1.5) can be seen as pseudo tasks, but with the important distinction that our synthetic tasks force features to be good at the specific problem of human action recognition rather than the more general problem of image processing.

Multi-task learning is a sub-domain of transfer learning [17, 77] in which the goal is to learn common representations for a number of related tasks. Other approaches consider feature selection in which a shared sparse set of features must be chosen for all tasks [3, 76], which is

similar to SVM learning with sparsity constraints [33]. Other work in multi-task learning uses neural network architectures to structurally force all the tasks to share intermediate representations [1, 17]. An approach by Rückert and Kramer [85] 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.

Unlike these sparsity based approaches, there is no explicit forced sharing of features in model recommendation; indeed, it is possible to recommend a model for a target task that is not shared with any other task. This distinction is important because multi-task learning is known to fail when the tasks jointly learned are insufficiently related to one another, and hence an open area of research in multi-task learning is how to select which tasks to learn together [41].

Given the popularity of boosting, it has unsurprisingly been applied to multi-task learning as well. These methods tend to follow the standard multi-task approach of enforcing sparsity, such as in Chapelle *et al*. [21] where boosting selects a common set of weights for weak learners across all tasks, and then individual tasks are allowed to sparsely deviate from that common weighting. Wang *et al*. [103] take a slightly different approach, where the sparsity is enforced by learning a partitioning (clustering) of the tasks, where all the tasks in a cluster are forced to share the same weights for the weak learners. Faddoul *et al*. [34] take yet another approach, in which the weak learners are joint classifiers of two tasks, and so the boosting naturally selects a compromise between two tasks. However, the limitation of their approach is that it does not easily scale to more than two tasks, and these techniques generally need closely related tasks.

The key difference is that our model recommendation method is completely agnostic to the type of model, and does not assume kernel learning or any other specific learning framework. Furthermore, model recommendation is less stringent about needing closely related tasks since there is no forced sharing (indeed, scenarios exist in which each user or task is recommended a different item).

2.4 Sharing Models: Selection from a Library

At the highest level of sharing, entire classifiers, algorithms, or models might be transferred among tasks. This relatively recent class of approaches is especially applicable when it is algorithms that must be shared, since selecting between black-box algorithms abstracts away the need to know anything about the underlying mechanisms of those algorithms.

For example, Aodha *et al*. [62] address the problem of selecting an appropriate optical flow algorithm from a library of four on a per-pixel level in a video, using a learned confidence measure. In the terminology of this thesis, the confidence measure can be seen as a rating which is used to pick the optical flow algorithm which is the highest rated at each pixel. They find that picking the "most confident" algorithm at each pixel (the highest rated) produces the best results, better even than the classifier trained to pick the algorithm. Of course, this result is not surprising from a collaborative filtering point of view: basing the recommendations on ratings alone has consistently been found to produce better recommendations than relying on (for example) movie metadata. What is interesting about the Aodha work is that the confidence measure does not need ground truth on the test video, and so can produce ratings even for unannotated data.

Aodha *et al*. [61] also have an earlier work in which they consider whole-clip classification

for choosing optical flow algorithms on a clip rather than pixel level.

Similarly, Cifuentes *et al*. [25] considers the problem of selecting a motion model for tracking feature points, where the motion model is to be selected from a library of six different models corresponding to the main directions of movement (left, right, forward, backward) as well as models for constant velocity motion and Brownian motion. For a particular video, the motion model is chosen by running a classifier on the video which predicts which motion model to use.

Conceptually related are approaches which attempt to gauge the quality of algorithms without having access to the underlying ground truth, as per Aodha *et al*. [62]. In this vein, Alt *et al*. [2] present a method for estimating the quality of templates for template tracking (how well they can be tracked) which is used to select between available templates in real-time.

Jammalamadaka *et al*. [40] present a method for estimating the quality of pose estimation without having the underlying ground truth. The underlying idea is that the output of successful pose estimation will have different statistical properties from unsuccessful pose estimation, and that this can be used to predict how good the pose estimation is without actually having the ground truth. However, unlike the Aodha *et al*.and Alt *et al*.works, this quality estimation is not yet used to select between competing algorithms.

2.5 Collaborative Filtering

Since our method is based on adapting collaborative filtering techniques to vision and robotics tasks, this section summarizes relevant work in the area.

An early work in collaborative filtering, Goldberg *et al*.'s Eigentaste algorithm [37] required users to rate all the items in a common "gauge" set of items to produce a fully complete ratings matrix over the "gauge" items. Then, this matrix is factorized (they relate the approach to PCA, but the underlying mechanics are essentially the same as the SVD based factorization) so that a user's rating of an item can be represented as a dot product between k dimensional hidden factors vectors associated with the particular user and item, which allows the prediction of each user's missing ratings. Equivalently, this can be seen as learning a linear regression from the ratings in the "gauge" set to the other items.

The reason for forcing a "gauge" set is that the problem of factorizing the ratings matrix is greatly simplified if all the ratings are known. However, modern approaches to the problem do not require any specific set of "gauge" items, but instead directly deal with the missing ratings.

Srebro and Jaakkola [96] demonstrated that the problem of matrix factorization under incomplete ratings is non-convex, unlike the complete ratings matrix scenario in which the well-known singular value decomposition (SVD) finds the optimal solution. Building on that work, Rennie and Srebro [82] presented a technique for maximum margin matrix factorization which has better factorization properties for incomplete and *discrete* ratings, like the discrete 1 to 5 star scale used in the Netflix prize. Although the objective function they propose is convex, it is difficult to implement the optimization, and so they further propose a gradient based method which is unfortunately non-convex, and as they say, "potentially bothersome".

Despite the non-convexity of the problem, gradient descent methods are still popular for computing the factorization. Wu [107] present several typical gradient descent based algorithms to factorization, as well as suggest that alternating least squares could be applied to the problem.

One of the best performing, and also most complicated to implement, methods is the Bayesian probabilistic matrix factorization of Salakhutdinov and Mnih [88]. In probabilistic matrix factorization methods, the missing ratings are assumed to be drawn from graphical-model-driven distributions governed by a number of "hyperparameters". These methods are sensitive to the choice of the regularization hyperparameters, and the contribution of Salakhutdinov and Mnih is to perform a second level of integration over possible hyperparameters by sampling using Markov chain Monte Carlo.

We primarily focus on the collaborative filtering approaches used by the BellKor team to (as a combined effort with two other groups) win the Netflix prize [7, 47]. As their 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 collaborative filtering, factorization techniques [30, 48] are in some sense the dominant paradigm, due to their good scaling properties and strong theoretical basis. This thesis mainly considers offline factorization with complete ratings, where the entire ratings matrix is available at once, but since the algorithms presented here are largely agnostic to the specific form of collaborative filtering used, on online matrix factorization technique like that of Mairal et al. [64] could be used in practice to extend these ideas to the case where the matrix is very large and continually growing through the addition of new tasks and models. Interestingly, collaborative filtering itself can be cast as a multi-task learning problem [79], where the goal is to learn common structures that allow item ratings to be predicted from other ratings. It should be stressed, however, that using multi-task learning to do collaborative filtering across a set of tasks is completely different from directly applying multi-task learning to those tasks.

Meta-data information has generally not been found to be especially useful in the Netflix prize, but the special case of the *time* when users assign ratings to movies has received substantial attention. This is because there are strong trends over time in people's movie preferences (*e.g.*, there are fads which wax and wane).

A common approach to temporal information is given by Xiong *et al*. [109] in which the factorization includes time factors, so that rather than predicting ratings as dot products between item and user factors, they are triple inner products of the user, item, and time factors: $r_{ijt} \approx \sum_{k=1}^{K} U_{ik} V_{jk} T_{tk}$. The method of Karatzoglou *et al*. [42] uses the same tensor factorization approach, but they call the third dimension "context" rather than time.

Although there is no reason to believe that temporal information has any relevance whatsoever to model recommendation, the multi-linear approaches developed to deal with that temporal dimension might likewise be applied to extending model recommendation to other dimensions of interest in particular applications.

Another major category of collaborative filtering is neighborhood methods, which might be viewed as variants on the well-known k-nearest-neighbor algorithm, where the neighbors are either computed across items (models) or users (tasks). That is to say, they predict the rating of an item according to how a user has rated similar items (an item-item neighborhood), or by how similar users have rated that item (a user-user neighborhood).

Since neighborhood methods can be computationally expensive, Chen *et al*. [22] use the fact that non-negative matrix factorization (NMF) can be seen as clustering to search only nearby clusters (according to the factorization) to the user/item rating to be predicted; the prediction is made by a simple weighted combination of the neighbors' ratings.

A sparse coding approach to collaborative by Szabó *et al*. [98] both learns the dictionary from which to perform the sparse coding, as well as solves for the sparse coding of each user (task) to that dictionary. This might be seen as a hybrid of a neighborhood approach and a factorization approach, in that the learned dictionary is akin to the factorization of the matrix, while the sparse coding is akin to finding a sparse set of neighbors.

Chapter 3

Applications and Datasets

The model recommendation method presented in this thesis is broadly applicable to a wide range of applications, but for the purpose of evaluating the core ideas, we concentrate on a few representative applications, each of which is carefully chosen to highlight a different facet of the problem.

3.1 Action Recognition

We consider use action recognition as a representative application for choosing between fully-trained classifiers in a classifier library.

"Action recognition" is a slightly misleading term, since in practice it usually refers to video clip classification. That is, given a video clip, the problem is to determine which of a fixed number of classes the video clip belongs to. In common datasets, the number of such classes (often referred to as "actions") varies from five to fifty. This application is representative of model selection problems where the library is composed of classifiers and the main objective is to improve the quality of the selection.

3.1.1 Common Low-level Features

The action recognition datasets all use libraries of classifier models trained on common low-level features, namely a typical STIP+HOG3D combination [45, 53], and a gridded histogram of optical flow (HOF). 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 $10\times10\times5$ pixel spatio-temporal cells, and a nine-dimensional HOF descriptor is computed for each cell in the standard way. So, for example, a $320\times240\times100$ video would be represented by a grid of $32\times24\times20\times9$ cells, and a typical trained model might by applied to a $12\times12\times10\times9$ scanning windows of the full grid. For computational efficiency, the optical flow is computed using the FlowLib [19] GPU-accelerated optical flow library. This representation is used primarily for the synthetic data, because it does not directly use appearance information.

The STIP+HOG3D features are a combination of STIP [53] and HOG3D [45] in a bag-of-words formulation. STIP features are detected in the video and described using HOG3D descriptors; these descriptors are quantized using k-nearest-neighbor to a codebook of 1000 "words", and the whole video is described as a histogram over the frequencies of these words. This representation is more powerful than the HOF representation, but can only be used on datasets where the appearance is meaningful (i.e., models trained on synthetic data using STIP+HOG3D will not work on real data, because the appearance between the synthetic and real data is so different).

3.1.2 UCF50 and UCF11

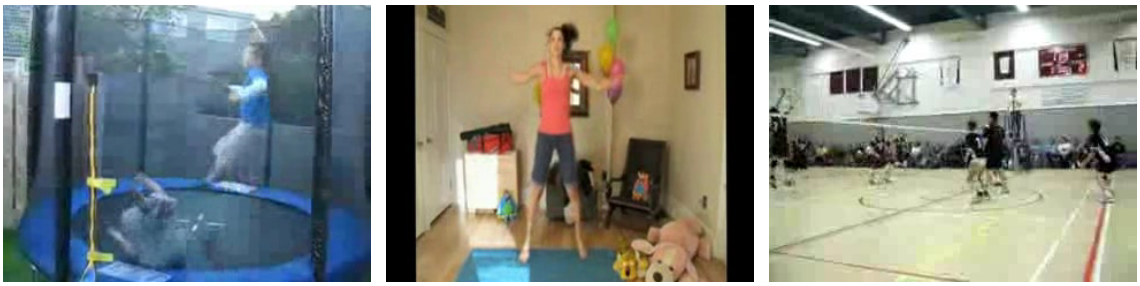


Figure 3.1: Sample frames from the the UCF50 dataset

The UCF "Actions in the Wild" datasets are common action recognition/classification benchmarks [59, 102]. Both are difficult datasets of videos of various actions harvested from YouTube; the dataset features large camera motions, compression artifacts, as well as large variations both in how actions are performed as well as in how they are filmed. The first dataset [59], which we call UCF11, had 1600 videos and 11 action classes. We primarily use this dataset in the early chapters on action recognition and feature seeding. That dataset's successor, commonly called "UCF50" [102], has approximately 6600 videos over 50 action classes. We use this dataset in the chapters on model recommendation.

Unless otherwise specified, for action recognition we use STIP [52] plus HOG3D [45] bag-of-words histograms as our low-level representation; this is commonly used as the foundation for action recognition systems and often performs similarly to more complex approaches [49].

We do not treat UCF50 as a monolithic 50-way classification problem as it is usually used, but instead partition it into a large number of tasks to evaluate our model recommendation ideas. This partitioning results in a large number of small tasks, each with highly limited training data (10.2 training samples on average, compared to the approximately 100 per action that would be available if UCF50 were treated as a single task).

UCF50 contains approximately 6600 videos, divided into 50 actions, with each action further subdivided into groups of videos, where each group comprises a set of (typically four) related videos. Since all the videos in a group are closely related, the intention of the dataset is that videos from the same group should not be used to both train and test. However, for our purposes (limited data, multi-task evaluation), we take advantage of the groups by treating a small set of groups as a "task", so that each task is (by itself) relatively "easy", but the problem is complicated

by the limited training data available for the task as well as the variation between each task and the other tasks in the database.

We produce each task by merging 1-3 groups of the same action, dedicating 2/3 of the videos in the merged set to the hidden test set, and the remaining 1/3 to visible training set for that task. This results in tasks with on the order of 2–10 positive training samples, and twice as many positive testing samples; We augment each task with an equal number of negative samples drawn at random from the other actions, so that each task is then a one vs. all binary classification problem with an equal number of positive and negative samples (so that chance is 50% accuracy). The mean number of training samples per task is 10.2, and the mean number of testing samples 24.3. Each group may be used in multiple training or test tasks (but there is no overlap between test and training).

We use the training groups to generate a library of 1000 classifiers trained on different groups, and also to generate a ratings store of those 1000 classifiers rated on 1000 tasks. Thus, the ratings store has size 1000×1000. Ideally, different data would be used to train the classifier library and build the ratings store, as rating a classifier against the same group from which it was trained results in an overly optimistic accuracy (rating) and possibly distorts the computed factorization.

3.1.3 UCF50 (ActionBank features)

This is the same UCF50 dataset, except the model library is not a library of trained classifiers, but rather the action templates which comprise the "bank" in the ActionBank feature representation [86]. ActionBank is composed of 200 different action templates, each of which is run over a video to produce a 71 dimensional feature, or $71 \times 200 = 14200$ dimensions in total. For the purpose of model recommendation, each bank entry (71 dimensions) is treated as a model, and the objective is to recommend the most discriminative bank entries for a particular action classification task.

3.1.4 Mind's Eye



Figure 3.2: Sample frames from the Mind's Eye dataset

This is another dataset for action recognition/classification. The Mind's Eye dataset consists of real videos of human actions provided for the DARPA Mind's Eye project [28]. Example frames from these videos can be seen in Fig. 3.2.

Tasks are generated from the Mind's Eye dataset as 1-vs.-1 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. “jump”. 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 action appears in multiple tasks, but the tasks differ in the selection of samples. Due to the limited amount of ME data, this dataset is not used to create its own ratings store, instead relying on ratings stores and model libraries generated from synthetic data for its recommendations. In particular, these four actions were chosen to align with actions available in the motion capture data (described in the next section). Using motion capture clips of the four actions, a model library of 180 1-vs.-1 classifiers is built, where each classifier is trained from 100 video clips, 50 from each of two actions chosen at random for that classifier (model).

3.1.5 Semi-Synthetic Rendered Motion-Capture



Figure 3.3: Sample frames from the the semi-synthetic action dataset

This is a dataset we have created by rendering videos of motion capture data from various viewing angles and with the addition of random distortions. This is novel, because there has been limited work on using synthetic data for action recognition. A number of approaches use synthesized silhouettes [23], or depth images [95], but these are applications where the synthesized data is very close to the real data (because silhouettes and depth images are comparatively easy to synthesize compared to full appearance).

This 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 [15]. Although such synthesis cannot yet produce photorealistic data, it has seen success when used for both depth [95] and motion [69] features.

A candidate pool of 10,000 models is generated by training classifiers on random pairs of synthesized actions. Similarly, 1000 synthetic tasks are produced as binary classification tasks between random mocap action pairs. The synthetic data uses approximately 50,000 videos to generate 10,000 models and 1000 tasks. Fig. 3.3 shows example frames from the rendered data.

For the production of a ratings store, a dense ratings store is produced by rating every model on every task in the database, using the accuracy on that task.

The test tasks are generated as the same type of synthetic tasks used to populate the matrix, using tasks that were held out from the matrix generation and factorization process. This dataset uses 180 test tasks, where each is divided into a training portion of 8–32 samples, and a larger testing portion.

Since it is difficult to produce synthetic data that is comparable to real-world data in terms of raw pixel-level appearance, we concentrate on the simpler task of generating synthetic data that matches real-world data in terms of *motion*. We make no attempt to mimic real-world appearance: the human model in our synthetic data is a abstract armature. However, in terms of motion it is a reasonable analog, since its motion is derived from human motion capture data (Fig. 5.1).

Motivation for synthetic data

The motivation for using synthetic data is threefold:

- Size: More synthetic data can be generated than is feasible to gather from real-life sources with any sort of annotation. For example, the largest video datasets may have approximately 5,000 clips, a number which can be synthetically generated in less than an hour.
- Detail: Available video datasets are very poorly annotated. The best case annotation is usually only a single class label for the entire clip, and in some datasets (Hollywood), even that is uncertain. In contrast, synthetic data can be generated with rich annotations, including per-pixel assignment to rigid bodies, foreground-background segmentation, and depth, in addition to the parameters (*e.g.*, motion capture file, position, and viewing angle) that were used to generate the motion and appearance.
- Variation: Bias is inherent in any dataset, simply because the 'world distribution' of possible video clips is impossible to sample from. Any dataset will be biased by the methodology used to collect it, and furthermore, even a supposedly ideal, unbiased dataset would not necessarily be the best in practice, since real world action recognition tasks are themselves 'biased'. But while eradicating bias is an unachievable goal, it is still worthwhile to build large and varied datasets, and synthetic data allows for automated variation generation in ways that are difficult to achieve with real datasets (for an analysis of dataset bias in real-world datasets, see [100]).

Synthetic data organization

The synthetic data is organized into groups of clips, or tasks. Each tasks consists of a number of *positive* samples all generated from a single motion capture sequence, and a number of *negative* samples randomly drawn from the entire motion capture dataset. In this way, each synthetic task represents an independent binary classification task where the goal is to decide which clips belong to the action vs. a background of all other actions. The actions used in the synthetic data do *not* necessarily correspond to the actions used in any final classification task on real data. Since the synthetic actions are randomly chosen out of motion capture sequences, they may not correspond to easily named actions at all. A typical synthetic clip might be 90 frames long, and a typical synthetic task might be associated with 100 such clips, corresponding to 50 positive samples and 50 negative samples.

A clip is produced by moving a simple articulated human model according to the motion capture sequence, with some added distortions. The synthetic data is rendered at a resolution of 320×240 and a nominal framerate of 30fps in order to match common datasets, such as MSR [111] and UCF50 [102]. The generation of this synthetic data is quite efficient, as actual rendering time is approximately 6× faster than realtime, so that two synthetic clips can be

rendered every second. At this unoptimized rate over 170,000 clips could be generated each day on a single computer; as such, the bottleneck with synthetic data lies in the processing of the resulting video.

Motion generation

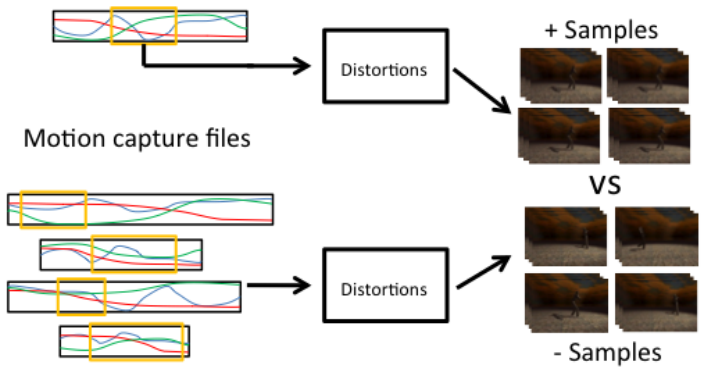


Figure 3.4: Generation of semi-synthetic data from motion capture files.

The motion of the human model in the synthetic videos is produced by taking motion capture sequences from the CMU motion capture database and adding temporal distortions and time varying noise to the joint angles (see Fig. 3.4 for a schematic representation of this process).

For each clip a motion capture file is chosen from the 2500 clips in the CMU motion capture database. If the clip is meant to be a positive example, then the motion capture file and approximate location within that file is given, and the starting frame is perturbed by approximately ± 1s. If the clip is meant to be a negative example, a motion capture file is randomly chosen from the entire database, and a starting position within that file is randomly chosen.

Next, temporal distortion is added by introducing a temporal scaling factor (*e.g.*, if the factor is 2.0, then the motion is sped up by a factor of two). Non-integral scaling factors are implemented by interpolating between frames of the motion capture file. Then, a random piece-wise linear function is used to dynamically adjust the temporal scaling factor of the rendered clip. In practice, we limit the random scaling factor to drift between a value of 0.1 and 2.0. Consequently, the timing of a rendered clip differs from that of the base motion capture file in a complicated and nonlinear fashion.

A similar approach is used to add time-varying distortion to the joint angles. A random piece-wise linear function is generated for every degree of freedom for every joint in the armature, and this function is simply added to the joint angles obtained from the motion capture sequence. The magnitude of this distortion is ± 0.3 radians.

We add several other distortions and randomizations to the synthetic data. The viewing angle is randomly chosen for each clip, as is the viewing distance. Additionally, the position of the actor/armature is randomized for each clip. The lighting is also randomized between clips, because the effects and positions of shadows can have a significant effect on the extraction of feature trajectories.

3.2 3D Scene Model Matching

The objective in 3D scene model matching is to estimate the 3D layout of a scene from a single image by matching that image to a library of full 3D meshes. This application is representative of applications where the main problem that model recommendation seeks to address is not the quality of the selection, but the computational cost. This application is especially constrained by computation costs, since the evaluation of each rating requires the rendering of a complex 3D scene from a new viewpoint. Fig. 3.5 illustrates this application. For this application, we use the dataset and technique of Satkin *et al*. [90].



Figure 3.5: The 3D scene matching application. Left: a 3D model from the Google sketchup warehouse rendered with normals (a "model"). Right: a real image of a room (a "task"). Center: the objective, to find the best matching 3D model for a given image. The match score or "rating" measures how well the rendered 3D normals correspond to the estimated surface normals in the image. Figure courtesy of Satkin *et al*..

3.2.1 Models

In this application, the models are literal 3D models of different room layouts. These models are harvested from the Google Sketchup warehouse; approximately 1500 models are obtained in this way, and then four rotations ($90\circ$ rotations) and a mirror flip are produced, so that each model from the Sketchup database produces 8 models in the library when these symmetries are taken into account. This means the library contains in the end approximately 12000 models.

3.2.2 Tasks

The tasks are *images* of rooms, in contrast with the 3D models that comprise the models. These are obtained from the SUN database [108]. There are a total of 216 room images, and there is no division between test and store tasks in this application because there are so few. Instead, results are computed using leave one out cross validation, holding one room image as the test task and the remainder as the store.

3.2.3 Ratings

The rating of a model (3D room layout) on a task (image of a room) is how well the 3D room layout in the model matches the image. That is, a perfect match would be if the 3D room model

were exactly the room depicted in the image. In practice, this match score is computed by determining the viewing position of the image, rendering the model from that viewpoint, and computing a score comparing the apparent surface normals in the visible image.

Note that this notion of rating implies that the surface normals are available in the 2d image (task), which is *not* the case in test tasks. Instead, in test tasks the rating is only approximate, because models are compared to the test image using *estimated* normals in the test image according to the classification method of Satkin *et al*. [90]; they have graciously provided their raw (and ground-truth) matching scores, so for this task we literally only see the ratings.

This gap between "true" ratings (computed from the ground-truth surface normals of the image) and "estimated" ratings implies that in this application a recommender system might also be able to find models that are better "true" matches than the models found even by testing every model in the library.

3.3 Skin Detection in Egocentric Video



Figure 3.6: Example detection results for different skin classifiers and video frames for the egocentric skin detection application. (Figure courtesy of Li and Kitani).

In this application the objective is to detect a person's arm (skin) in a first-person video. The data and classifiers were provided by Cheng Li and Kris Kitani [55]. As with the 3D model matching application, in this application, reducing the computational cost by limiting the number of models that must be evaluated is an important concern. The challenge in this application is that detecting skin on a per-pixel level is greatly complicated by the constantly changing illumination experienced by a person in motion. Example frames and detection results for Li and Kitani's classifiers in this task can be seen in Fig. 3.6. Currently, Li and Kitani use detectors trained for the same person as the test tasks, albeit not from the same frames. Both the detectors and tasks are drawn from the same long (18000 frames) video sequence, but the detectors are trained from the first 1000 frames and the remaining frames held out for testing. Ground-truth annotations are provided by human annotators, an example of which can be seen in Fig. 3.7.



Figure 3.7: The egocentric skin detection application. Left: a frame from an egocentric (chest or head mounted) video. Center: The ground truth skin mask to be detected. Right: The detection results of detector 23 from the library.

3.3.1 Models

Models in this application are per-pixel skin detectors (classifiers). A library of 100 models is trained from the first 1000 frames in the video, where the frames have been clustered according to their HSV color-space histograms to produce clusters of frames with assumed similar illumination, with one model trained per such a cluster. Each frame is 640×480 in resolution. Each model is a per-pixel skin classifier, implemented as a decision tree forest on top of various color and texture descriptors of a patch around the pixel of interest, including SIFT, HOG, BRIEF, and ORB descriptors.

In the provided dataset, the training data is for the same person as the testing data so that the models vary in the illumination conditions they are trained for but not the underlying skin tone they are meant to find. However, our recommendation technique could easily accommodate models which were tuned not only for illumination, but for skin tone as well.

3.3.2 Tasks

Each task is a frame of ego-centric video, *i.e.*, a largely normal frame. Although the wide-angle view of the camera introduces geometric distortions, these are not of significant concern for the per-pixel detectors which are the models. The test tasks are sampled every 50 frames; thus, there are 17000 / 50 = 340 test tasks. Due to the limited sampling of ground truth, the ratings store is produced by leave-one-out cross validation (that is, for each test task, consider all the other tasks to be the store). While not ideal, the large temporal distance between tasks (50 frames, > 1 second) limits the similarity between adjacent tasks that would otherwise be a concern for overfitting.

3.3.3 Ratings

The rating of a model (skin detector) on a task (image) is the F1 score (harmonic mean of precision and recall) of its skin classification for that frame. However, just like in the 3D model matching application where these "true" ratings are not necessarily available on a real application (because it would require having the ground truth), for this application there are "estimated"

ratings which are emulated by adding noise to the ground-truth ratings.

That is, we take the ground-truth F1 scores and add noise. This is meant to emulate the result of estimating the quality of the classification without having the underlying ground truth, by some automated method. This notion of rating the detection without knowing the ground truth is not as outlandish as it seems; for example, recent work by Jammalamadaka *et al*. [40] demonstrates how in pose estimation "good" poses have different statistics from badly estimated poses, and that this can be used to rate the quality of pose estimation without having the ground truth. Thus, we simulate such an approach as simply being the true F1 score with some normally-distributed error.

3.4 Robot Controller State-Machines

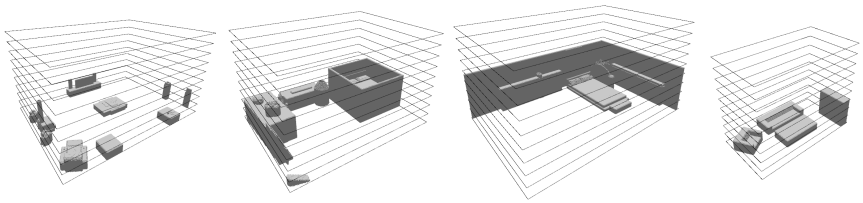


Figure 3.8: Example room layouts obtained from the Google Sketchup library

This objective of this application is to choose the best state machine to control a robot in a room-coverage task. The robot runs a simple state machine and the objective is to cover (vacuum) as much of the floor of a room as possible within a fixed time limit.

Since the state-machines are non-deterministic, and the starting location of the robot in the room is randomized, each time a state machine is rated (run), it produces a different rating. Therefore, ratings are not static, but can be re-evaluated to improve the statistical estimate of the true mean rating. This application is representative of applications where the tasks are actual processes where a model can be rated multiple times, and where the objective is to find an exploration-exploitation compromise between searching for a better model and using the best found model so far.

Floor-plans are produced from the Google Sketchup database of 3D models by finding models of furnished rooms and computing a rasterized 2D grid representation of the traversable floor area, where each grid cell represents a square inch. A typical room might be 300×180 grid cells (25ft$\times$15ft). Although these are not strictly real floor plans, they are designed by humans and likely share the same general features as the arrangement of furniture within real rooms (See Fig. 3.8).

3.4.1 Simulator and State-Machine Controllers

This application considers a simple robot operating in a 2d room layout. The robot is modeled as a disk with a 7 inch radius which can drive forward and backward (subject to maximum velocity constraints) and turn (the robot can turn completely in place), subject to maximum turning rate

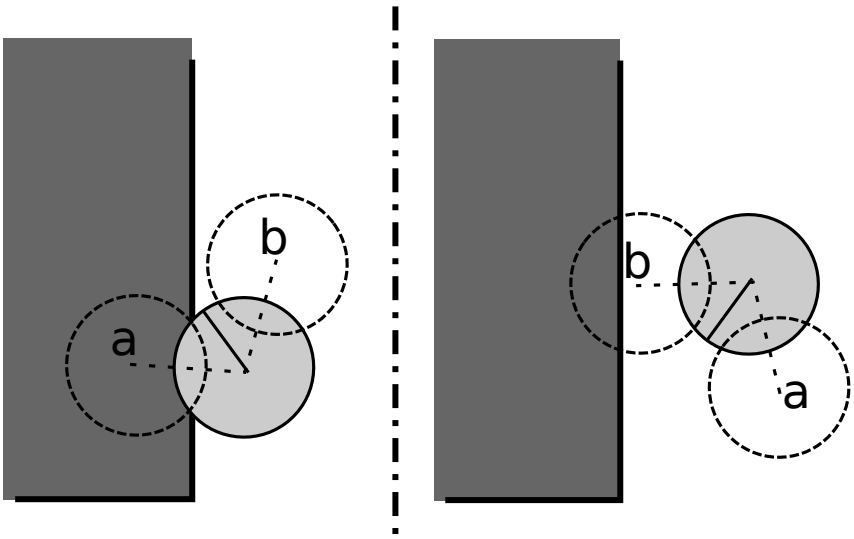


Figure 3.9: Simulated robot sensors. Left: collision and bump sensor a are activated. Right: only bump sensor b is activated.

constraints. The robot is assumed to operate continuously, and a grid cell is 'covered' (vacuumed) if any part of the robot ever touches it. The robot cannot drive through furniture.

The robot has three types of sensors: it can detect when it has directly hit something (a collision), it has a number of 'bump' sensors, and it has timers. The 'bump' sensors are implemented as detecting whether an object of the same size as the robot would collide at a fixed relative location to the robot (see Fig. 3.9). Each timer senses whether a fixed amount of time has elapsed in the current state.

The robot runs a simple state machine. At each simulator tick t, the robot starts in some state s_t. The robot then evaluates all its sensors (collision, bump, and timers). Let E_t be the set of sensor events that occurred in the tick; a static transition table governs the state transitions, so that $T(s_t, e)$ returns the possible set of new states the robot could take after observing event e in state s_t. Note that there can be multiple outgoing transitions from a single state-event pair: in this case, the robot chooses one of the new states randomly with uniform probability. Since multiple events might fire in a single tick, let the total set of possible new states be the union of all the possible transitions, $P_t = \bigcup_{e \in E_t} T(s_t, e)$. Then, the robot randomly picks a new state from P_t.

Each state is associated with a fixed output 'instruction'. There are two classes of instructions: 'drive' and 'spiral'. In drive mode, the output specifies the linear velocity and turning rate, as fractions of the max (possibly negative). The maximum linear velocity of the robot is 14 inches/s and the maximum turning rate 12 rad/s. In spiral mode, the robot's linear velocity is fixed, and the turning rate is governed by the equation $v_\theta = \frac{v_0}{(\delta t+1)^{1/2}}$, which will drive the robot in an Archimedean spiral whose direction and spacing is governed by the output parameter v_0; δt is the amount of time elapsed in the current state. See Fig. 3.10.

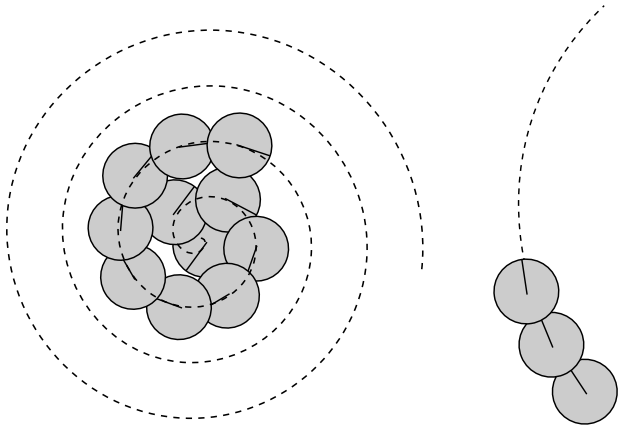


Figure 3.10: Simulated robot drive modes. Left: "spiral" (robot drives at constant velocity in an Archimedean spiral), right: "drive" (robot drives at constant velocity and turn rate, producing a circular arc).

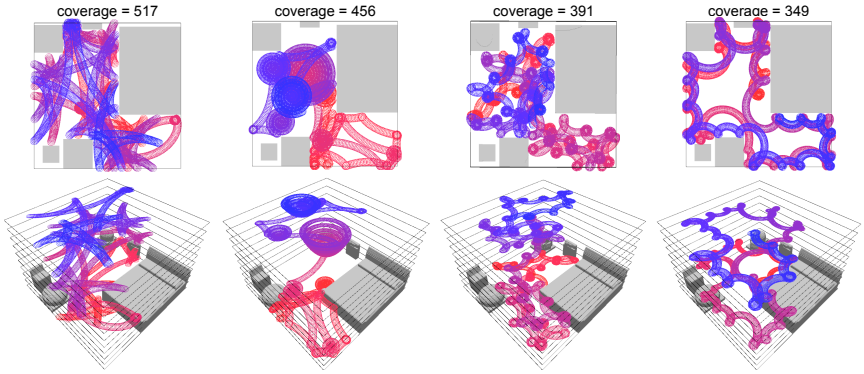


Figure 3.11: Examples of different state machines running on the same room, one run of each state machine viewed from both overhead (top row, color is time: red→blue) and perspective (vertical axis and color are time). Despite the simplicity of the state machine representation, a wide range of behaviors is possible, and the interaction of these behaviors with room layouts is difficult to characterize analytically.

3.4.2 Tasks

Out of the approximately 1500 floor plans obtained from Google Sketchup, we use approximately 1000 as the set used to build the database of coverages (the ratings store), and hold out 526 as the testing set.

3.4.3 Models

The models in the library are state machines. Given this state machine definition, a library of 200 state machines is generated by picking 200 rooms from the training set, and for each room trying 1000 random state machines and retaining the best for the library.

Despite the simplicity of this state machine representation, there is a wide range of possible behaviors for the robots, a few of which are exhibited in Fig. 3.11.

3.4.4 Ratings Store

A ratings store is built by running each of the 200 models in the library on the 1000 training set rooms for ten iterations to produce a 200×1000 ratings database.

Chapter 4

Action Recognition

Action recognition is used as a recurring example application throughout this thesis, and so this chapter gives background on the action recognition application and presents our work on action recognition as a reference system for how these methods typically function. In that respect, although the work presented here was one of the first to use tracked interest point trajectories and to integrate spatial information into action recognition, in terms of overall architecture the presented method is entirely orthodox, and "bag of words" (BoW) methods such as it are still very popular in the community.

Action recognition is often means video clip classification (*e.g.*, classify a video clip into one of a fixed number of known semantic categories), and the bulk of approaches to this problem rely on reducing each video clip down to a known length descriptor, and then learning a machine-learning based classifier (such as the ever-popular support vector machine [SVM]) to classify videos into the classes according to their descriptors.

Our method is based on trajectory features and a pairwise relationship descriptor of the same, and has two stages. The first stage is a typical bag-of-words method built on quantized trajectory (interest points tracked with KLT) descriptors. The second stage takes those quantized interest points and derives a spatial relationship descriptor between them. For the final classification, the histogram of trajectory words and the spatial relationship descriptor are simply concatenated and fed into a support vector machine.

This descriptor is produced by estimating all of the cross probabilities for descriptor labels, that is, for each pair of labels and each action a relative location probability table (RLPT) is built of the observed spatial and temporal relationships between descriptors of those labels under the given action. Then, any descriptor label can compute its estimate of the distribution over action probabilities using the trained cross-probability maps. These estimates are combined for each descriptor label, and the final descriptor vector is presented to a classifier.

Figure 4.1 shows a visualization of the spatial relationship descriptor that our method uses to classify video clips into action classes.



Figure 4.1: Pairs of descriptors probabilistically vote for action classes; pairs voting for the correct action class are shown in yellow, with brighter color denoting stronger (more informative) votes. For "answerPhone", the relative motion of the hands is particularly discriminative. These results employ trajectory fragments on the Rochester dataset [71], but our method works with any localized video descriptor.

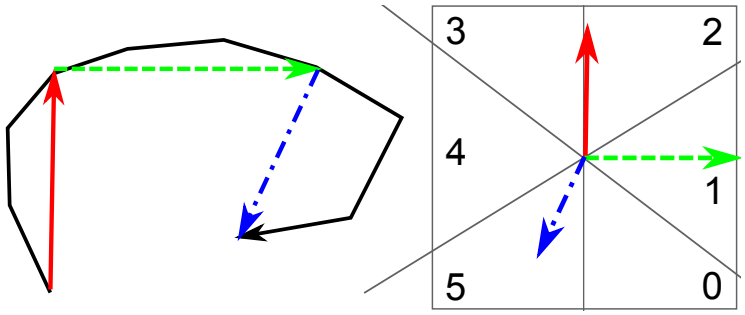


Figure 4.2: Sequencing code map (SCM) quantization breaks a trajectory fragment into a number of stages (in this case three) that are separately quantized according to a map (in this case a 6-way angular map). These per-stage labels, called sequence codes, are combined into a final label for the fragment, which in this case would be $215_6 = 83_{10}$.

4.1 Base Descriptor: Trajectons

Our method for action recognition starts with interest points tracked with KLT through the video. Then, in order to formulate the method in a bag-of-words framework, the tracked interest points are quantized into a fixed number of trajectory "words", or "trajectons" as we termed them in [67].

Taking inspiration from Messing *et al.*'s approach to quantizing trajectories [71], we quantize the trajectories using a derivative table similar to Messing *et al.*'s, which we call a sequencing code map (SCM), examples of which can be seen in Figure 4.3. However, rather than using quantized derivatives to look up probabilities, we simply combine the quantized indices over the fixed length trajectory fragment into a single label encoding quantized derivatives at specific times with each fragment (see Figure 4.2).

For SCM quantization, each trajectory fragment is divided into k consecutive stages of length t frames, such that $kt \leq T$, where T is the total length of the fragment. The total motion, or

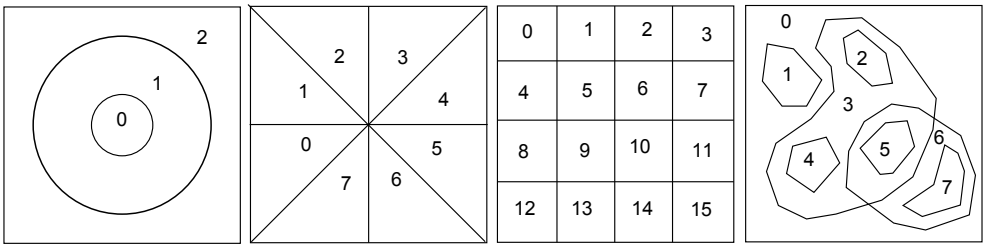


Figure 4.3: Examples of possible sequencing code maps (SCMs) or relative location maps (RLMs). Our experiments focus on angular maps (second from left) but our method is completely general.

summed derivative, of each stage is computed as a $(dx, dy)_k$ pair for each stage. This (dx, dy) vector is quantized according to the SCM into one of n stage labels, or sequence codes; the k sequence codes are combined to produce a single combined label that can take on k^n values. For example, with 3 stages and an SCM containing 8 bins, there would be 8^3 or 512 labels total. Since the time to quantize a stage is a single table lookup regardless of how the table was produced, this method is extremely computationally efficient (*i.e.*, the computation time does not grow with an increasing number of quantization bins).

Formally, we denote by $M(dx, dy)$ the SCM function, which is implemented as a two-dimensional lookup table that maps from a dx, dy to an integer label in the range of 0 to $n - 1$ inclusive. We denote by $(dx, dy)_k$ the derivative pair for stage k. Then the assigned label is given by $l = \sum_{j=0}^{k} n^j \cdot M(dx_j, dy_j)$.

Besides the convenience of not having to build a codeword dictionary and the reduced computational cost, our introduction of this quantization method is meant to demonstrate that the pairwise descriptor does not depend on data-driven clustering techniques. The quantized labels produced by SCM quantization are unlikely to correspond nicely to clusters of descriptors in the dataset (*e.g.*, parts), yet the improvement produced by our pairwise relationships persists.

4.2 Alternate Base Descriptor: STIP-HOG

In addition to augmenting our own trajecton descriptors with pairwise relationships, we also consider Laptev *et al.*'s space-time interest points (STIPs) [53], in conjunction with Histogram of Oriented Gradient (HOG) descriptors. This descriptor/descriptor combination has achieved state-of-the-art performance on a variety of video classification and retrieval tasks. A variable number of STIPs are discovered in a single video and the local space-time volume near each interest point is represented using an 72-dimensional descriptor. These HOG descriptors are quantized using a codebook (typically pre-generated using k-means clustering on a large collection) to produce a discrete label and a space-time location (x, y, t) for each STIP.

4.3 Augmenting Descriptors with Pairwise Relationships

The second stage of the method builds a spatial relationship descriptor on top of the quantized interest points from the previous stage.

While the previous discussions focused on STIP and trajecton descriptors, our method for pairwise spatio-temporal augmentation applies equally well to any type of descriptor that can be quantized and localized. In the remainder of this section, a "descriptor" is simply the tuple (l, x, y, t): a codeword label in conjunction with its spatio-temporal position. The set of all observed descriptors is denoted F.

4.3.1 Pairwise Discrimination with Relative Location Probabilities (RLPs)

Starting with the observation that Naïve Bayes is a linear classifier in log space, in this section we formulate a pairwise representation first in the familiar terms of a Naïve Bayes classifier, and then demonstrate how to expose more of the underlying structure to discriminative methods.

We start with the assumption that all pairs are conditionally independent given the action class. Then, if descriptors are quantized to L labels and the spatial relationships between descriptors are quantized to S labels, we could represent the full distribution over pairs of descriptors with a vector of L^2S bins. Unfortunately, for even as few as $L = 100$ trajectory labels and $S = 10$ spatial relationships, there would be $100^2 \times 10 = 100,000$ elements in the computed descriptor vector, far too many to support with the merely hundreds of training samples typically available in video datasets.

This descriptor vector must be reduced, but the direct approach of combining bins is just equivalent to using a coarser quantization level. Instead, taking inspiration from the Max-Margin Hough Transform [66] and Naïve Bayes, we build probability maps of the spatial relationships between descriptors, and instead of summing counts, we accumulate probabilities, allowing pairs to contribute more information during their aggregation.

Specifically, we produce a descriptor vector B of length AL, where A is the number of action classes (*e.g.*, walk, run, jump, *etc.*), and where each entry $B_{a,l}$ corresponds to the combined conditional probability of all pairs containing a descriptor with label l given the action class a. In other words, a bin contains a descriptor label's probabilistic vote for a given action class, and we could compute a Naïve Bayes estimate of the probability of all the observed descriptors given an action by summing all the votes for that action: $\log P(F|a) = \sum_{l \in L} B_{a,l}$. However, instead of summing these in a Naïve Bayes fashion, we present the vector as a whole to discriminative machinery, in our case a linear SVM. We now describe how we accomplish this descriptor vector reduction.

Notation. Formally, a video segment has a number of quantized descriptors computed from it. A descriptor $f_i \in F$ is associated with a discrete quantized label $l_i \in L$ as well as a spatio-temporal position (x_i, y_i, t_i) indicating the frame and location in the frame where it occurs. For a pair of descriptors within the same frame and a given action $a \in A$, there is a probability of the two descriptors occurring together in a frame $P(l_i, l_j|a)$ as well as the relative location probability (RLP) for their particular spatial relationship $P(x_i, x_j, y_i, y_j|l_i, l_j, a)$. We make the simplifying assumption that RLPs depend only on the relative spatial relationship between the

two descriptors, so that $P(x_i, x_j, y_i, y_j | l_i, l_j, a) = P(dx, dy | a, l_i, l_j)$, where dx and dy are slight abuses of notation that should be understood to mean $x_i - x_j$ and $y_i - y_j$ where appropriate. This assumption enforces the property that the computed relationships are invariant to simple translations of the descriptor pairs.

Probabilistic formulation. The reduction to a descriptor vector of length AL is done by selectively computing parts of the whole Naïve Bayes formulation of the problem. In particular, a full probabilistic formulation would compute $P(F|a)$ and select the a that maximizes this expression. Since Naïve Bayes takes the form of multiplying a number of descriptors' probabilities, or in this case pair probabilities, we can exploit the distributive and commutative properties of multiplication to pre-multiply groups of pair probabilities together, and then return those intermediate group probabilities rather than the entire sum. This can be seen as binning the pair probabilities.

Assuming descriptor pairs are conditionally independent, we can compute the probability of a descriptor set F given an action a according to the equation

$$P(F|a) = \prod_{f_i \in F} P(l_i|a) \prod_{f_j \in F} P(l_j|l_i, a) P(dx, dy|a, l_i, l_j), \tag{4.1}$$

which strictly speaking double-counts pairs since each pair is included twice in the computation; however, since we are only interested in the most likely action, this is not an issue.

In practice, we employ log probabilities both to avoid issues with numerical precision from extremely small values and to formulate the problem as a linear classifier. In this case the log probability expression becomes

$$\begin{aligned} \log(P(F|a)) = &\sum_{f_i \in F} \log(P(l_i|a)) + \\ &\sum_{f_j \in F} \log(P(l_j|l_i, a)) + \log(P(dx, dy|a, l_i, l_j)). \end{aligned} \tag{4.2}$$

To simplify the expression we assume uniform probabilities for $P(l_i|a)$ and $P(l_j|l_i, a)$. Later we can include nonuniform label probabilities by simply concatenating the individual label histogram to the pairwise descriptor vector when both are presented to the classifier. Thus, our probability expression becomes

$$\begin{aligned} &\log(P(F|a)) = \\ &\quad \sum_{f_i \in F} \sum_{f_j \in F} \log(P(dx, dy|a, l_i, l_j)) + C, \end{aligned} \tag{4.3}$$

which is simply a formal way of stating that the whole log probability is the sum of all the pairwise log probabilities. Since we are only interested in the relative probabilities over action classes, we collect the uniform probabilities for labels into a constant C which does not depend on the action a, and which is omitted from the following equations for clarity. We now wish to divide this expression into a number of sub-sums that can be presented to a classifier, and this expression leaves us a great deal of flexibility, since we are free to compute and return sub-sums in an arbitrary manner.

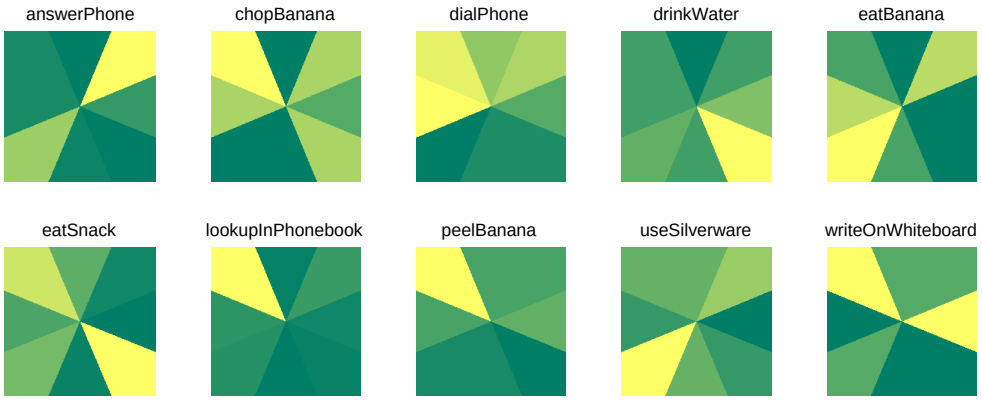


Figure 4.4: Relative location probabilities (RLPs) for descriptor labels 25 and 30 over all actions in the Rochester dataset, using an 8-way angular map. Lighter (yellower) indicates higher probability. We see that for the answerPhone action, descriptors with label 30 tend to occur up and to the right of descriptors with label 25, whereas for useSilverware, descriptors with label 30 tend to occur down and to the left of those with label 25.

Discriminative Form. We rewrite Equation 4.3 in such a way as to bin probabilities according to individual descriptor labels. In particular, we can rewrite it in log form as

$$\log(P(F|a)) = \sum_{l \in L} \log(P(b_l|a)), \tag{4.4}$$

where

$$\log(P(b_l|a)) = \sum_{f_i \in l} \sum_{f_j} \log(P(dx, dy|a, l, l_j)). \tag{4.5}$$

The expression $\log(P(b_l|a))$ is the bin probability, which directly corresponds to an element of the descriptor vector according to $B_{a,l} = \log(P(b_l|a))$. Since there are A actions and L labels, this B vector contains AL elements.

4.3.2 Estimating Relative Location Probabilities from Training Data

The previous section assumed that the relative location probabilities (RLPs) were simply available. However, these probabilities must be estimated from real data, requiring some care in the representation choice for the relative location probability tables (RLPTs). An RLPT represents an expression of the form $\log(P(dx, dy|a, l_i, l_j))$, where a, l_i, and l_j are considered fixed. In practice this means that it must represent a function from a (dx, dy) pair to a log probability, and that we must represent one such function for every (a, l_i, l_j) triplet of discrete values. While many representations are possible, we use an approach similar to that used for staged quantization.

We denote by $M(dx, dy)$ a function that maps a (dx, dy) pair to an integer bin label, allowing n such labels. We refer to this map as the relative location map (RLM), possible forms of which can be seen in Figure 4.3. Then the RLPT for a given (a, l_i, l_j) triplet is a list of n numbers, denoted T_{a,l_i,l_j}. An RLP can then be retrieved according to:

$$\log(P(dx, dy|a, l_i, l_j)) = T_{a,l_i,l_j}[M(dx, dy)]. \tag{4.6}$$

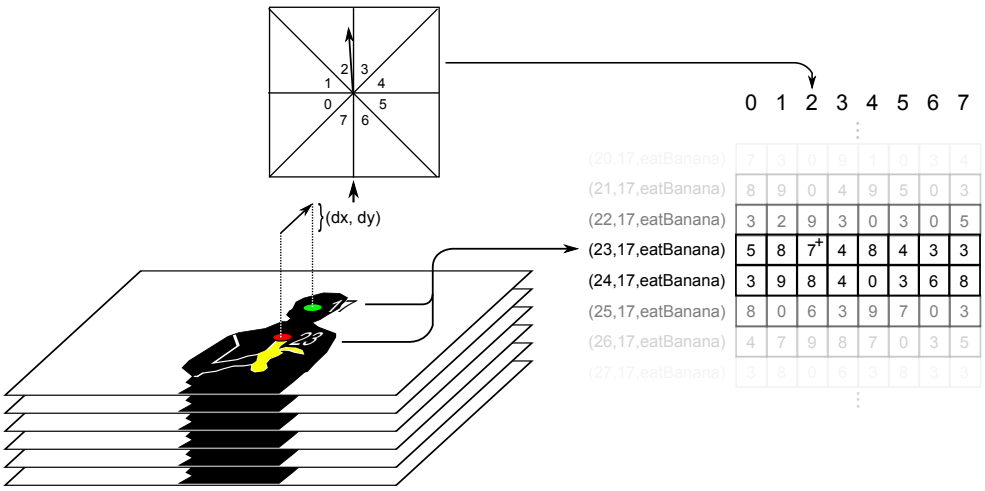


Figure 4.5: descriptors with labels 17 and 23 are observed in a frame of a training video of the class eatBanana. The descriptor with label 17 has a relative displacement of (dx, dy) from that with label 23, which maps to bin #2 in an 8-way angular RLM. Thus, we increment bin #2 in the corresponding table entry; these counts are all converted to estimated log probabilities after the entire video collection is processed.

For example, with 216 labels, 10 actions, and 8 bins in the RLM, storing all the RLPs would require $(216)(10)(8) = 3,732,480$ entries in 466,560 tables.

Estimating the RLPTs is simply a matter of counting the displacements falling within each bin (see Figure 4.5), and finally normalizing by the total counts in each map. Since some bins may receive zero counts, leading to infinities when the log probability is computed, we use a prior to seed each bin with a fixed number of pseudo-counts. Examples of RLPTs found in this way can be seen in Figure 4.4.

This method could seemingly generate very sparse probability maps where most bins receive few or no real counts. However, in practice almost all of the bins receive counts. A typical situation (in this case our experiments on Rochester's Daily Living dataset) might have 10 classes, 216 descriptor labels, and a probability map with 8 bins, for a total of $(216^2)(10)(8) = 3.7 \cdot 10^6$ bins. For Rochester we have approximately 120 training videos, each of which is approximately 600 frames long. If we track 300 descriptors per frame, and only consider in-frame pairs, then across all videos we will have $(300^2)(600)(120) = 6.5 \cdot 10^9$ pairwise descriptors. Thus, on average each bin will receive over a thousand counts.

4.3.3 Extension to Temporal Relationships

The method is naturally extended to include temporal relationships. Rather than representing the relationship between two descriptors as a (dx, dy) pair, it is represented as a (dx, dy, dt) triple. The RLPT then contains entries of the form $\log(P(dx, dy, dt|a, l_i, l_j))$, which are indexed according to a mapping function $M(dx, dy, dt)$, so that

$$\log(P(dx, dy, dt|a, l_i, l_j)) = T_{a,l_i,l_j}[M(dx, dy, dt)]. \tag{4.7}$$

Previously, the map M could be stored as a simple image, whereas with spatial-temporal relationships this map is a volume or series of images. When counts are accumulated or probabilities evaluated, only pairs of descriptors within a sliding temporal window are considered, since considering all pairs of descriptors over the entire video would both result in a prohibitively large number of pairs and prevent the method from being run online on a video stream. Nevertheless, the change from considering only pairs within a frame to pairs within a temporal window vastly increases the number of pairs to consider, and depending on the number of descriptors generated by a particular descriptor detector, it may be necessary to randomly sample pairs rather than considering them all. We find that for STIP-HOG we can consider all pairs, while for SCM-Traj we must sample.

4.3.4 Classification

We train a linear SVM [20] to classify video clips, which is a straightforward matter of presenting computed B vectors and corresponding ground truth classes from training clips. Each bin in B can be interpreted as a particular label's vote for an action, in which case the classifier learns the importance of each label's vote.

Since, when considered in isolation, pairwise relationships are unlikely to be as informative as the base descriptors from which they are derived, we present a simple method for combining the raw base descriptor histograms with the computed pairwise log probability vectors. We do not present this combination method as the canonical way of combining the two sources of information, but rather as a convincing demonstration that the proposed pairwise relationships provide a significant additional source of information rather than merely a rearrangement of the existing data.

Supposing that H represents the histogram for the base descriptors, and B represents the computed pairwise relationship vector, then one way of combining the two would be to simply concatenate the two vectors into $[H, B]$, and present the result to a linear SVM. However this is unlikely to result in the best performance, since the two vectors represent different quantities. Instead, we separately scale each part, and then simply cross validate to find the scaling ratio p that maximizes performance on the validation set, where the combined vector is $[pH, (1-p)B]$. This scaled vector is simply presented to the SVM.

4.4 Evaluation

We evaluate our pairwise method on the UCF11 (the original, 11 class version, not the 50 class UCF50 dataset used later in this thesis) and Rochester Daily living datasets (see Chapter 3). To evaluate the contribution of our method for generating pairwise relationships, we consider two different types of base descriptors: the trajectory based descriptors we introduced earlier, and Laptev *et al.*'s space-time interest points. We consider both our discriminative formulation (denoted D-pairwise) and a Naïve-Bayes formulation (NB-pairwise) for our pairwise descriptors, where the NB-pairwise results are primarily intended as a baseline against which to compare.

Table 4.1 summarizes our results. Our experiments are designed to evaluate the effect of adding spatial and temporal relations to the descriptors and to understand in detail the effect of

Table 4.1: Action recognition accuracy on standard datasets. Adding pairwise descriptors significantly boosts the accuracy of various base descriptors.

Method	UCF11	Rochester
STIP-HOG (single) (Laptev *et al*. [53])	55.0%	56.7%
STIP-HOG (NB-pairwise alone)	16.4%	20.7%
STIP-HOG (D-pairwise alone)	46.6%	46.0%
STIP-HOG (single + D-pairwise)	59.0%	64.0%
STIP-HOG-Norm (single) (Laptev *et al*. [53])	42.6%	40.6%
SCM-Traj (single)	42.3%	37.3%
SCM-Traj (NB-pairwise alone)	14.3%	70.0%
SCM-Traj (D-pairwise alone)	40.0%	48.0%
SCM-Traj (single + D-pairwise)	47.1%	50.0%

various parameters on the performance of the augmented descriptors. Clearly, significantly more tuning and additional steps would go into building a complete, optimized video classification system. In particular, we do not claim that our performance numbers are the best that can be obtained by using complete systems optimized for these data sets. We use the evaluation metric of total accuracy across all classes in an n-way classification task.

On both datasets we use 216 base descriptor codewords for both trajectories and STIP-HOG. The number 216 results from the choice of three stages with a 6-way mask for the staged quantization ($6^3 = 216$), and we use the same number for STIP-HOG to make the comparison as even as possible. Likewise, for both datasets we use an 8-way spatial relationship binning for the RLPTs. Combined results are produced by cross validating on the scaling ratio.

The UCF11 dataset was chosen for its difficulty, in order to evaluate the performance of pairwise relationships outside of highly controlled environments. In particular, this dataset is challenging because the videos contain occlusions, highly variable viewpoints, significant camera motion, and high amounts of visual clutter.

On UCF11 we find that discriminative pairwise descriptors are not as informative as the base descriptors, which is not unexpected since the diversity of the dataset means there are unlikely to be strong, consistent relationships between descriptors. Nevertheless, we still find modest gains for combinations of pairwise and individual descriptors, on the order of 5%. This means that the pairwise descriptors are providing an additional source of information, rather than just obfuscating the information already present in the individual descriptor histograms. The Naïve Bayes pairwise evaluation performs poorly, but better than chance.

Furthermore, we can see that the performance of our simple fixed quantization on trajectories performs similarly to normalized STIP-HOG descriptors, but significantly worse than non-normalized STIP histograms. This suggests that much of the discriminative power of STIP descriptors might originate from the variation in the quantity of descriptors found in different videos.

On Rochester we observe that the pairwise descriptors for STIP-HOG do not perform as well as the individual STIP-HOG descriptors, but that the combination outperforms both, which is consistent with the results for UCF11. For trajectory descriptors, the pairwise descriptors alone

Table 4.2: Action recognition accuracy with temporal relationships on UCF11

Method	STIP-HOG	Traj-SCM
NB-Pairwise (baseline)	16.4%	14.3%
NB-T-Pairwise	22.2%	31.2%
D-Pairwise (baseline)	46.6%	40.0%
D-T-Pairwise	49.2%	39.7%

significantly outperforms the base descriptors, a reversal from UCF11. The combination of the two outperforms both the individual and pairwise, but adds only a modest gain on top of the pairwise performance.

For both types of descriptors, the gains with pairwise relationships in combination are much larger than for UCF11, which is explained by the greater consistency of spatial relationships between codeword labels due to the fixed viewpoint and highly consistent actions. Qualitatively examining which pairs contribute to a correct action identification supports this hypothesis: as can be seen in Figure 4.1, the pairwise descriptors supporting an action appear to be strongly tied to that action. For STIP-HOG, the Naïve-Bayes pairwise formulation once again performs poorly, however for trajectories the Naïve-Bayes pairwise is the strongest performer. This suggests that for some applications, even simple relationships can give very good performance.

4.4.1 Effect of Temporal Relationships

The results with using spatial and temporal relationships on UCF11 are shown in Table 4.2 in which X-T-Pairwise denotes the classifier (discriminative or Naïve-Bayes) augmented with temporal relations. For these results, we have used the same 8-way spatial relationship binning combined with a 5-way temporal binning, for a total of 40 bins. The pairwise relationships are evaluated over a 30 frame sliding window. For STIP-HOG, all pairs within the window are considered, but for trajectories we sample 1/20 of the pairs in the interest of tractability. Note that even with this sampling, a four second UCF11 clip can produce over 100,000,000 pairs when using trajectory descriptors.

The performance of the discriminative pairwise relationships remains virtually unchanged for Traj-SCM, but there is a modest performance boost for STIP-HOG. The Naïve-Bayes versions continue to perform worse than the discriminative ones, however the temporal relationships have a much larger impact on their performance. The difference is especially dramatic for NB-Pairwise vs. NB-T-Pairwise with STIP-HOG, where the temporal relationships have more than doubled the accuracy from 14.3% to 31.2%.

4.4.2 RLPT Sparsity

Earlier we argued that the relative location probability tables should not be sparse based on a simple counting argument. Empirically, we find that for the Rochester dataset 71.6% of the entries receive counts, and that 91.7% of the tables have at least one count in one of the 8 bins.

The number of tables containing at least 100 counts is 41.1%, and 15.2% of tables have over 1000 counts. These numbers validate our original claim that the tables are not sparse.

4.5 Conclusions

This chapter has presented a typical action recognition system, complete with typical limitations. Although this method is in some sense relatively simple, it already requires substantial effort in obtaining training data. Worse, since it is very statistic-driven, the learned relationships in the RLPT and the weights in the SVM classifier are highly dataset specific, and the final trained classifier is unlikely to generalize well to another dataset. Given how sensitive the method (and these methods in general) are to the unique patterns in each dataset, it is difficult to improve performance on a given dataset without having more data for exactly that dataset.

In the following chapter, we attempt to alleviate this problem by using a number of synthetic tasks to find *generally good* low-level features to replace the SCM-quantization used in this chapter. If this method is a subsistence farmer toiling in isolation, feeding only himself, then the next chapter seeks to build a community of farmers to decide which crops (features) are the best overall.

Chapter 5

Feature Seeding

The previous chapter introduced a typical action recognition system in which almost everything was learned anew on each dataset, with the only thing shared between datasets being the fixed underlying action recognition algorithm itself. In contrast, model recommendation seeks to share (potentially) entire trained classifiers between different tasks and datasets. This chapter presents a method which we call "feature seeding", in which low-level features are shared (chosen) according to their aggregate classification performance when applied to a pool of semi-synthetic (motion-capture-derived) action classification tasks.

The feature seeding method presented in this chapter can be seen as a transitional form between the completely isolated learning of Chapter 4 and the collaborative method of model recommendation. In some sense, feature seeding can be interpreted as model recommendation without the feedback of the ratings on the current task: that is, what should be recommended to a user or task which has not rated any items in the library? In that case, the recommendation system has to fall back to a "generally good" recommendation, an average best. Feature seeding explores this idea by using a store of synthetic data from which to determine which features are the average best, as well as exploring different notions of what constitutes the "average best".

In full model recommendation, because there *is* the feedback of the ratings on the new task, a much higher level of sharing is possible than in this restricted case. Indeed, in the case of full model recommendation entire classifiers can be shared, because the system is recommending specifically good classifiers. But in this restricted situation where the objective is to find generally good features, what does a generally good classifier mean? Certainly no single classifier could be chosen which would work well across all tasks.

For the feature seeding scenario, there must be some level of flexibility in the task; that is, the shared elements cannot be the final classifiers, but should be some lower-level component so that the final learning can take place on the new task.

In particular, the action recognition method of the previous chapter had two stages: in the first, trajectory descriptors were quantized to a codebook using a human-designed quantization scheme, and it is this first stage of quantization that feature seeding seeks to replace.

Note that this replacement is not specific to the action recognition system of the previous chapter; many popular bag of visual words (BoW) techniques rely on quantizing descriptors (where a descriptor might be a trajectory fragment, or a HOG descriptor computed around an interest point, or many other possibilities) computed from video; generally either simple un-

supervised techniques such as k-means clustering [31, 53, 75, 92] or hand-crafted quantization strategies (such as our earlier work on trajectons [67] and sequencing code maps [68]) are used. The end result of these quantization schemes is a histogram that counts how frequently features are quantized into particular quantization bins. We generalize this notion of quantization by suggesting that each bin of the resulting histogram can be considered an independent feature defined by a classifier that decides whether a given descriptor should be counted by that bin. Then the problem of designing a *quantization scheme* can be seen as the problem of recommending a set of such histogram count classifier features.

We consider the problem of choosing *generally good* features for human action recognition using synthetic data. More concretely, in this chapter we consider the problem of recommending a quantization scheme for a bag-of-words technique, using a corpus of synthetic data as the source from which to recommend the quantization scheme. We use a simple form of feature selection in which we independently recommend features by rating them on all the synthetic datasets, and then assigning them aggregate ratings from the ratings on individual synthetic datasets.



Figure 5.1: Our synthetic data looks nothing like real data, but in terms of motion they are similar.

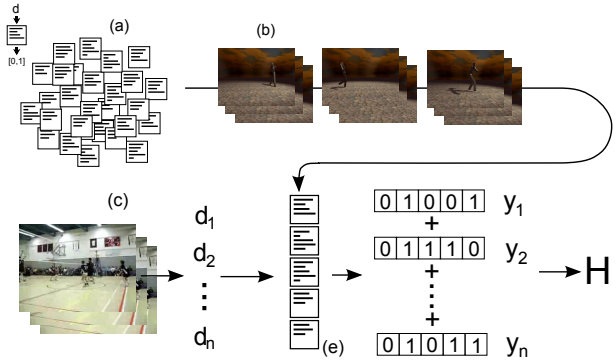


Figure 5.2: System overview: a pool of randomly generated features (a) is filtered, or seeded, on synthetic data (b) to produce a greatly reduced number of features (e) that are likely to be informative. Real data (c) has descriptors (*e.g.*, trajectories) extracted, and these descriptors are fed through the seeded feature set to produce label vectors y_i, one per descriptor. These label vectors are then accumulated into the histogram H, which represents the video clip.

5.1 Overview

The basic organization of the method can be seen in Figure 5.2. First, a set of synthetic video clips is generated using motion capture data. These clips are generated in groups (datasets), where each group is an independent binary classification problem; this is described in detail in Section 3.1.5.

Next, raw motion descriptors are extracted from the synthetic data pool in the form of trajectory snippets [68, 71] and histogram of optical flow (HOF) descriptors around space-time interest points (STIP) [53]. We use the unquantized form of the trajectory descriptors, since our purpose is to replace the SCM quantization scheme of the previous chapter with one recommended from synthetic data. Likewise, we use the 90-dimensional STIP-HOF descriptors without any pre-emptive quantization.

Each clip produces many descriptors– trajectory descriptors produce on the order of 300 descriptors per frame of video, while STIP-HOF produces closer to 100 descriptors per frame. These descriptors are sampled to produce a candidate pool of features, where each feature is associated with radial basis function classifier whose support vectors are randomly drawn from the descriptors. Then the synthetic data is used to rank features based on their aggregate classification performance across many groups of synthetic data. We denote the highly-ranked features selected in this way as the *seeded* features. The seeded features can then be applied to real data and used as input to conventional machine learning techniques. For evaluation, we consider the seeded features in a basic BoW framework, using linear SVMs as classifiers.

Note that for this work we are not proposing a complete action recognition system, but concentrate only on motion features in order to evaluate the potential of such a screening based approach from synthetic data.

5.2 Feature Pool Generation and Evaluation

We consider a pool of candidate features, where intuitively each feature can be viewed as a Gaussian radial basis function classifier. Formally, each feature evaluates a function of the form

$$f_k(d) = \text{clip}(\sum_i w_{k,i} \cdot e^{-\beta_{k,i}||d-v_{k,i}||^2}), \tag{5.1}$$

where $v_{k,i}$ is one "support vector" for the feature f_k, and $w_{k,i}$ and $\beta_{k,i}$ are the weight and beta for that support vector. The clip$(.)$ function clips the value to the range $[0, 1]$, so that values less than zero (definite rejections) are thresholded to zero, while values above one (definite accepts) are thresholded to one and all other values are unchanged.

We choose to use features of this form because RBF can approximate arbitrary functions [93], and random classifiers and features have, by themselves, shown benefits over other representations [16, 81]. The choice of each feature as an independent classifier means that, after features have been seeded, only the selected features need to be computed on the test dataset.

The feature can also be seen as computing an intermediate representation $q(d)$ corresponding to a descriptor d, so that

$$q(d) = (f_1(d), f_2(d), \ldots, f_K(d)) \,. \tag{5.2}$$

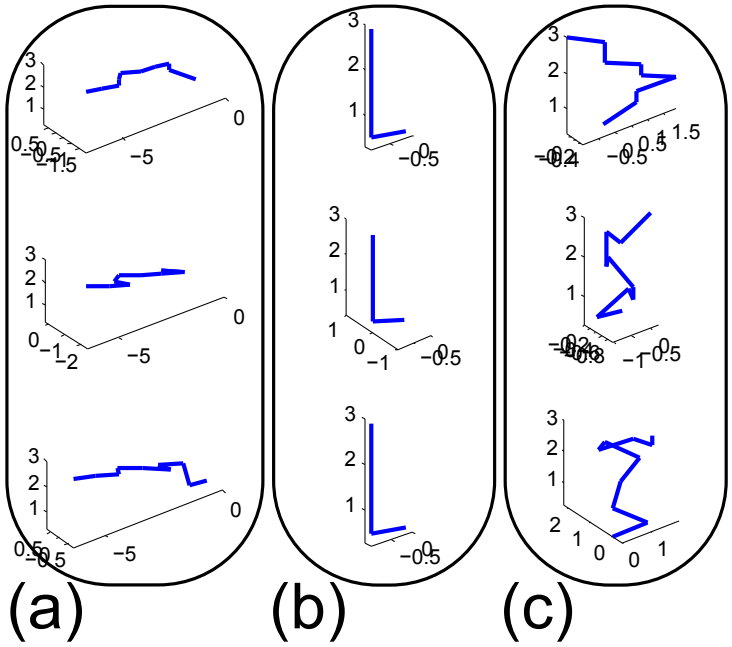


Figure 5.3: Examples of trajectory descriptors accepted by different classifier features. Some features represent simple concepts, such as leftward movement (a), or a quick jerk (b), while others do not correspond to anything intuitive (c). Given limited labeled data (c) could be indicative of overfitting. Feature seeding allows us to confidently determine that the chosen features generalize well.

When the pool of features is evaluated, the "histogram" bin corresponding to a feature is evaluated according to

$$b_k(D) = \sum_{d \in D} f_k(d), \tag{5.3}$$

where D is a set of descriptors (*e.g.*, all the descriptors computed from a given video). The entire histogram is expressed as

$$h_D = (b_1, b_2, \ldots, b_K) = \sum_{d \in D} q(d), \tag{5.4}$$

which is to say that the feature f_k is treated as an indicator function for whether a descriptor belongs to label k, where a descriptor might have multiple labels, and where the labels a descriptor d_i takes on are given in the vector q_i.

Given this feature definition, a pool (in this case, of size 10,000) of such features are generated by randomly selecting sample vectors from the synthetic dataset's descriptors to be the "support vectors" of the features in the pool. The weight associated with each sample vector is chosen from a normal distribution $N(0, 1)$, and the β associated with each sample vector from a uniform distribution over the range $[0, 10]$. These parameters were chosen arbitrarily to generate a large range of variation in the classifiers. Example trajectory descriptors that might be accepted by these types of features can be seen in Fig. 5.3.

5.3 Feature Seeding/Filtering

Given the pool of features, the method selects for, or seeds, a good set of features from the pool by rating them on a set of synthetic data. In practice, the seeding is similar to a single iteration of boosting, with the important difference that the seeding attempts to find features that work well across many different problems, rather than a single one.

Let P_n and N_n correspond to the sets of descriptor sets (videos) in the positive and negative sample sets, respectively, of a synthetic group n. Then we can express the rating $r_{k,n}$ of a feature k on group n ($n = 1, \ldots N$) as

$$r_{k,n} = \max_t \frac{\sum_{D \in N_n} I(b_k(D) \leq t) + \sum_{D \in P_n} I(b_k(D) > t)}{||N_n|| + ||P_n||}, \tag{5.5}$$

where $b_k(D)$ is the result of evaluating feature k on descriptor set (video) D, and $I(.)$ denotes the indicator function.

Note that this is just the accuracy of a decision stump on the $b_k(D)$ values. We have also considered using mutual information to compute ratings, but we find that it has slightly worse performance, probably because the stump-classifier accuracy we use here is a better match for the final SVM classification. However, our method does not depend on any single rating metric, and it is straightforward to swap this metric for another.

Now, we express the aggregate accuracy of a feature over all groups as

$$A_k = g(\{r_{k,n} | n = 1, \ldots, N\}), \tag{5.6}$$

where $g(.)$ is a function that operates on a set. In our case, we consider three possible aggregation functions g: $g_{\text{min}}(X) = \min(X)$, $g_{\text{max}}(X) = \max(X)$, and $g_{\text{avg}}(X) = \text{mean}(X)$. Intuitively, g_{min} takes the worst-case performance of a feature against a collection of problems, g_{max} takes the best-case performance, and g_{avg} takes the average case. Note that because the evaluation problems are randomly generated from a large motion capture database (see Section 3.1.5), it is unlikely that they will share any action classes in common with the target task. The goal is to select features that perform well against a variety of action recognition tasks (*i.e.*, that can discriminate between different human actions).

Then we simply rank the features according to their A_k values and select the top s ranked ones. In practice, we use seeding to select the top $s = 50$ features from a pool of 10,000.

Given a set of training and test videos on real data, we compute histograms h_D, where each histogram is computed according to (Eqn. 5.4) over the reduced set of s features. Then we simply train a linear SVM as the classifier.

5.4 Evaluation

We primarily evaluate on the UCF11 dataset (the original 11 class version, not the 50 class UCF50 used later in this thesis) as described in Chapter 3. In addition, we evaluate this work on the Microsoft Research Action Dataset (MSR) [111], which consists of of sixteen relatively long (approximately 1000 frames per video) videos in crowded environments. The videos are taken from relatively stationary cameras (there is some camera shake). The dataset only has

three actions — clap, wave, and box, with each action occurring from 15 to 25 times across all videos. The actions may overlap. For evaluation we consider MSR to be three separate binary classification problems, i.e., clap vs. all, wave vs. all, and box vs. all, rather than a three-way forced choice because the actions overlap in several parts. Each problem has an equal number of negative samples drawn by randomly selecting segments that do not feature the action in question, so for example, the wave vs. all problem is a binary classification between the 24 positive examples of the wave action and 24 negative examples randomly drawn from the videos. Due to the limited amount of data in this set, evaluation is by leave-one-out cross validation.

As described earlier, for feature seeding we use a synthetic dataset, which consists of 2000 short videos; the "actions" in this dataset do not necessarily correspond to any of the action classes in either the UCF11 or MSR datasets.

5.4.1 Feature Statistics

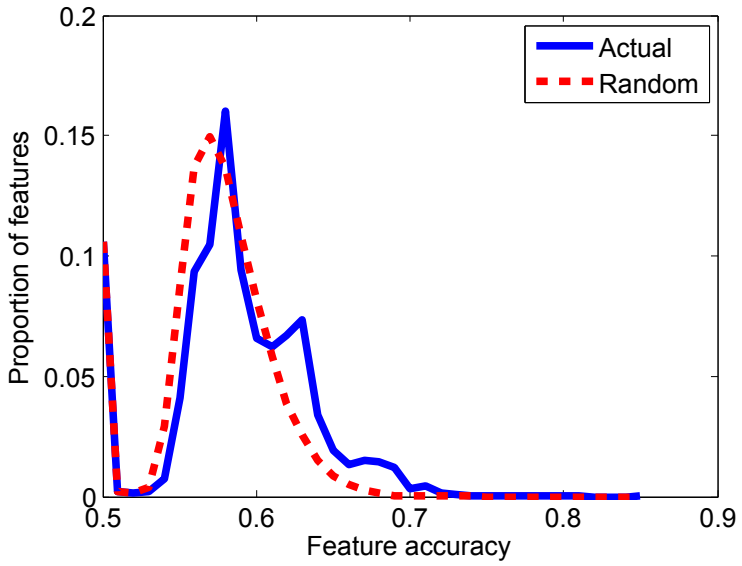


Figure 5.4: Accuracy distribution of RBF classifier features on synthetic data, compared with the expected number of false positives. Above accuracy 0.61, the majority of features are true positives. The difference between these two distributions is statistically significant to $p < 0.001$ according to the Kolmogorov-Smirnov test.

A natural question to consider is how informative these RBF features are; that is, how likely is our seeding method to find useful features. Because the features are evaluated by treating them as stump classifiers, the worst an individual feature could do is 0.5 accuracy; any lower, and the classifier simply flips direction. Since there is noise in the data, a classifier that is uncorrelated with video content can still vary in value across videos, and this means that it is possible for it to obtain an accuracy better than 0.5 on the limited data simply by chance. If we were considering a single feature, then we could ignore this unlikely possibility, but with a pool of 10,000, statistically we can expect to see many such false positives.

Table 5.1: Results on the UCF11 dataset (motion features only).

Method	Total accuracy
Seeded RBF [STIP] (g_{max})	34.4
k-means [STIP]	36.6
k-means [Traj] (MSR centers)	36.6
k-means [Traj] (synth centers)	37.0
Seeded RBF [STIP] (g_{min})	38.6
Seeded RBF [Traj] (g_{max})	38.9
Seeded RBF [Traj] (g_{avg})	39.4
Unseeded RBF [Traj]	40.2, $\sigma = 1.9$
Trajectons [Matikainen *et al.* [68]]	42.2
All 10,000 RBF [Traj]	46.0
Seeded RBF [Traj] (g_{min})	46.0

Table 5.2: Comparison of Seeding Source on UCF11.

Method / Source	UCF11	Synthetic
Seeded RBF [Traj] (g_{max})	38.6	38.9
Seeded RBF [Traj] (g_{avg})	34.7	39.4
Seeded RBF [Traj] (g_{min})	41.0	46.0

It is easy to empirically estimate the false positive distribution by simply randomly permuting the labels of all of the test videos; in this way, a classifier cannot be legitimately correlated with the video labels, and the resulting distribution must be entirely due to false positives.

As can be seen in Fig. 5.4, the accuracy distribution of the real features is quite different from the false positive distribution. In particular, the real feature distribution is shifted to the right of the random distribution, indicating that there are more high-accuracy features than would be expected by chance, even in the worst-case scenario that the vast majority of the features are uninformative. Note that the false positive distribution takes on a log-normal type distribution, albeit with a spike at 0.5 corresponding to zero-variance features (in practice, it is easy to reject these features, even on real data, since they are features which return the same output for every input). The same test performed with the aggregation techniques produces similar results, indicating that the aggregation techniques also reveal informative features.

5.4.2 Comparison with Other Quantization Methods

Since the goal of the proposed technique is to improve on the early quantization and accumulation steps of the bag-of-words model, a natural baseline against which to compare is the standard bag-of-words model consisting of k-means clustering followed by nearest neighbor quantization and histogram accumulation. Additionally, for the UCF11 dataset we compare against the somewhat more sophisticated SCM quantization technique we described in the previous chapter [68].

Our results on the UCF11 dataset are shown in Table 5.1. Here the feature shows large gains over both k-means and random feature subsets, at 46.0% to 37.0% and 40.2% respectively. Additionally, feature seeding improves upon our SCM quantization technique, which obtains an accuracy of 42.2%. The power of our technique is further emphasized by the fact that feature seeding uses only 50 features compared to the 216 of the SCM trajectons. Even with a pairwise spatial relationship coding scheme, that technique achieves an accuracy of 47.7%, which is only slightly better than the performance of the seeded features without any spatial information.

Note that our performance with 50 seeded features matches that of running the entire candidate set of 10,000 features. Beyond the obvious computational and storage benefits of processing only 50 features instead of 10,000, methods that build on top of these quantized features will likely benefit from the reduced dimensionality (*e.g.*, if pairwise relationships are considered, it is better to consider 50×50 rather than 10000×10000). While the "kitchen sink" approach of feeding all 10,000 classifiers into an SVM worked in this case (likely due to the resilience of linear SVMs against overfitting), other classifiers (*e.g.*, decision trees, randomized forests) may not be as robust.

The results of this comparison on the MSR dataset are shown in Table 5.3. Overall, the feature selection posts relatively large gains in the g_{max} and g_{avg} selection methods, while g_{min} remains largely the same as for k-means. For the individual classes, the selection method improves performance on the clap and box categories, while performance on wave is largely similar.

It is interesting that the selection techniques that perform well are exactly inverted between MSR and UCF11, with g_{max} and g_{avg} performing well on MSR, while g_{min} performs well on UCF11. In practice, g_{avg} works like a weaker g_{max}, so it is unsurprising that its performance is similar to that of g_{max} on both datasets. Between g_{min} and g_{max}, however, we suspect the difference is due to how similar the datasets are to the synthetic data that was used for feature selection. The MSR dataset is much more similar to the synthetic data than the UCF11 dataset, which may explain why the more aggressive g_{max} selection performs better on the former while the more robust g_{min} selection performs best on the latter. More specifically, the MSR dataset has a fixed camera and simple human motions, which matches the cinematography of the synthetic data (albeit varying in the specific actions). By contrast, UCF11 exhibits highly variable cinematography and includes non-human actions (*e.g.*, horses and dogs) as well as actions with props (*e.g.*, basketballs and bicycles).

5.4.3 Comparison of Base Descriptors

The results of a comparison of base descriptors (trajectories vs. STIP-HOF) is shown in Table 5.1. Overall, the performance of STIP-HOF features is worse than that of trajectory-based ones. However, note that the best selection method (g_{min}) outperforms k-means for both STIP and trajectory features, and that g_{min} outperforms the other two methods for both features.

Comparison to Unseeded RBF Features

As an additional baseline we compare the performance of the features seeded from the synthetic data to that of random sets of features. The purpose of this baseline is to establish whether the gains seen with the classifier sets over k-means are due to the selection process, or whether the

Table 5.3: Seeding outperforms k-means, unseeded RBF, and boosting on MSR.

Method	Clap	Wave	Box	Total
Boosting (50/500)	65.0	60.0	57.0	60.7
Unseeded RBF	60.0	61.0	61.7	61.0
Seeded RBF (g_{min})	60.7	62.5	60.4	61.2
k-means (MSR centers)	53.6	62.5	68.7	61.6
k-means (synth centers)	53.6	62.5	68.7	61.6
Boosting (50/10000)	75.0	64.6	52.0	63.9
Seeded RBF (g_{avg})	71.4	62.5	66.7	66.9
Seeded RBF (g_{max})	75.0	58.3	70.8	68.0

classifier-based features are inherently more informative than k-means histogram counts. As can be seen in Tables 5.1 and 5.3, the performance of random classifier sets is very similar to that of codebooks produced by k-means, indicating that random classifier sets are by themselves about only as powerful as k-means codebooks. It is only after selection (either on the data itself, if there is enough, or on synthetic data) that significant gains are seen over the k-means baseline.

However, despite the similar performance of unselected classifier features to k-means features, there is reason to believe that the classifiers themselves are a better choice for selection than k-means based features. On both MSR and on UCF11 the performance of k-means features derived from vastly different datasets (synthetic data vs. MSR) is very similar. Furthermore, experiments on synthetic data have suggested that the actual choice of k-means centers has only a small effect on performance. As a result, it is difficult to produce large gains by optimizing for k-means centers.

Comparison with Feature Selection on Real Data

We perform experiments using AdaBoost for feature selection on the MSR dataset (see Table 5.3). While boosting on the data itself improves performance on the clap action, the overall performance increase is modest, suggesting that when features are selected from the entire pool of 10,000 classifiers, boosting overfits. When the features are boosted from smaller subsets chosen at random, the overall performance is closer to that of unseeded features. However, the average performance of boosting on the real data is not much better than that of random subsets, and lower than that of seeded features.

Next, we evaluate the contribution of the synthetic data itself, in order to rule out the possibility that it is only the seeding technique (*i.e.*, randomly partitioning the data into groups and then evaluating aggregate performance) that produces performance gains. We perform our feature seeding using the real training data as the seeding source. In order to mimic the structure of the synthetic data groups (one action class vs. everything else), we partition the UCF11 training data into groups, where each consists of one action class vs. the remaining 10. We further randomly partition each group into five, for a total of 55 groups. We then perform the feature seeding. These results are shown in Table 5.2. Note that for every selection method (*e.g.*, g_{min}), the seeding from synthetic data outperforms the seeding from the real data. Additionally, the

selection method $g_{\min}$ is the best regardless of the seeding source. Thus, the synthetic data itself plays an important role.

5.5 Conclusion

This work demonstrates that even simple selection/recommendation mechanisms can give a boost in performance from a corpus of dissimilar, but related, datasets. More tellingly, features selected from synthetic data have better performance than those selected from the real data, despite the similar sizes of the datasets, indicating that the synthetic data itself contributes to the success of the technique. There are many possible reasons for this: the synthetic data may have more variation, it might help prevent overfitting, or it may simply be cleaner (*i.e.*, there is no clutter or camera motion in the synthetic data, so it may select features that better represent the core human motions than those selected from the noisy real data). These results are promising, since they suggest that using a corpus of even crudely related labeled data can help on action recognition tasks.

The main drawback of this method is that including more synthetic data does not improve performance. The likely reason is that the simple aggregate statistics by which the features are selected (*i.e.*, mean performance) converge quickly, and so past the point of convergence, additional data cannot offer an improvement. Another possible reason is that additional synthetic starts to overfit the features to the particular *global* biases of the synthetic data. The discovery of this limitation was one of the main motivations for the development of the model recommendation framework: rather than attempting to find features that are generally good across *all* the synthetic data, in model recommendation we use the corpus of datasets to tailor features *specifically* for the target dataset.

Chapter 6

Single Model Recommendation

This chapter describes the basic model recommendation problem which forms the foundation on which the extensions are built: namely, the problem of recommending a *single* model from the library for a specific task, or equivalently, of predicting the ratings of all the models in the library *independently* (*i.e.*, disregarding possible interactions between models).

This method is conceptually straightforward and the most direct equivalent to consumer item recommender systems. The method starts with a database of ratings (the *ratings store*). This database can be seen as a matrix where rows correspond to models, and columns correspond to tasks. This chapter assumes that in the store every model has been rated on every task (the matrix is complete); this assumption of a *dense* ratings matrix is relaxed in Chapter 8.

Then, given a new target task and a subset of rated models on that task (the probe models and associated probe ratings), collaborative filtering techniques are used to predict the ratings of all the models, and return the model with the highest predicted rating as the recommendation.

However, since in a model recommendation problem the ratings are *not* ground truth, but *noisy estimates* of the "true" ratings, it is theoretically possible that a recommender system could produce predicted model ratings which are closer to the "true" ratings than the noisy estimates. In other words, it may be possible for a recommender system to produce a better selection than if even every model were tried and the one with the highest apparent rating (accuracy) chosen. As this chapter will show, this is not a theoretical oddity, but an empirical effect that manifests across datasets and applications.

6.1 Estimated vs. Ideal ratings

One of the fundamental difficulties facing any feature selection or recommendation system is that the performance of features, as measured on a *training* set of data, may differ, perhaps significantly, from the performance of those features on the *test* data. For example, a set of features might be chosen that (through overfitting) perform perfectly on the training data; however, such features are unlikely to preserve that performance on the test data. This is especially likely when the number of training samples is small. We term the ratings of features on the training data *estimated* ratings, since they are noisy estimates of how we expect those features to perform on the actual test data. Likewise, we denote the actual performance of features on the test data to

be the *ideal* or *true* ratings. If features were all totally independent, then the best estimate of the ideal ratings would be produced by the estimated ratings from the training data. However, if features are related, then potentially a model of feature relationships could be used, along with the estimated ratings, to better predict the ideal ratings. For example, if the performance of features were related by a lower-dimensional linear model, then knowledge of that model could be used to reduce the error of the estimated ratings by constraining them to that linear model.

This distinguishes our feature recommendation problem from more typical recommender systems with explicit feedback, where ratings (generally given by human users) are taken as ground truth. That is to say, in a typical recommender system, the system cannot know better than the user what his or her rating of an item is; if the system and the user disagree, the system is in error.

6.2 Collaborative Filtering Algorithms

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 [48], 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 [48]: First, estimate the global mean

$$\mu = \frac{\sum_i \sum_j r_{i,j}}{mn}.$$

Then, estimate initial factors as

$$\phi_i = \frac{\sum_j (r_{i,j} - \mu)}{n},$$

and

$$\psi_j = \frac{\sum_i (r_{i,j} - \mu)}{m}.$$

A second iteration produces the final estimates:

$$\phi_i = \frac{\sum_j (r_{i,j} - \mu - \psi_j)}{n},$$

and

$$\psi_j = \frac{\sum_i (r_{i,j} - \mu - \phi_i)}{m}$$

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

$$\psi_t = \frac{\sum_{i \in P} (r_{i,t} - \mu - \phi_i)}{|P|}, \tag{6.1}$$

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. There are many possibilities for this factorization [48, 82, 96], but for clarity here we present the singular value decomposition (SVD) version. Formally,

$$\bar{R} = F^T D, \tag{6.2}$$

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

$$(\hat{F}^T)x = \bar{r}_p \tag{6.3}$$

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:

$$\bar{r}' = F^T x, \tag{6.4}$$

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 αs are the neighbors of the target task, and the prediction is a weighted combination of the neighbors.

Simple Neighborhood

A simple neighborhood technique is to simply find the k nearest neighbors of a task (by either correlation coefficient or Euclidean distance) and predict the ratings for that task as the mean of the neighbors' ratings of the models. This can work surprisingly well.

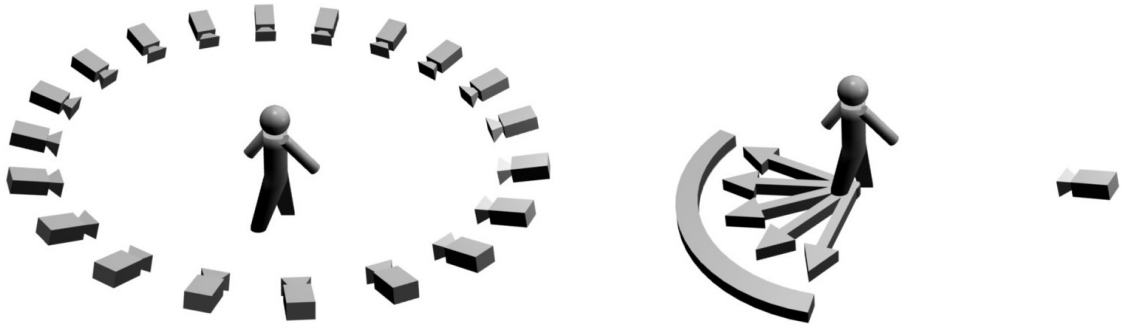
Algorithm 1 k-Nearest-Neighbor Collaborative Filtering

function PREDICTEDRATING(C, m_i)
  $N \leftarrow \text{Neighbors}(C, R, k)$
  return $r_p = \frac{\sum_{q \in N} r_{iq}}{|N|}$
end function

6.3 Interpretations of Factorization-Based Recommendation

Generally, the motivation for factorization has been the good scaling properties and empirical performance, but in addition to those practical considerations, factorization approaches have several intuitive interpretations that shed light on the recommendation problem in ways that (for example) neighborhood techniques do not. This section explores some of these intuitive interpretations in order to explain some of the interesting behaviors of recommendation, as well as to justify the use of factorization approaches.

6.3.1 Factorization as Finding Latent Factors



(a) Models (b) Tasks

Figure 6.1: 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.

This interpretation springs directly from interpreting the factorization prediction as a dot product between a model *factors vector* and a task factors vector, so that

$$\bar{r}_i j = \hat{F}_i \cdot D_j, \tag{6.5}$$

where $\hat{F}_i$ and D_j refer to the ith and jth columns of the respective matrices. That is, the factorization can be seen as finding k latent factors for each model and task.

As an illustrative example, 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 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

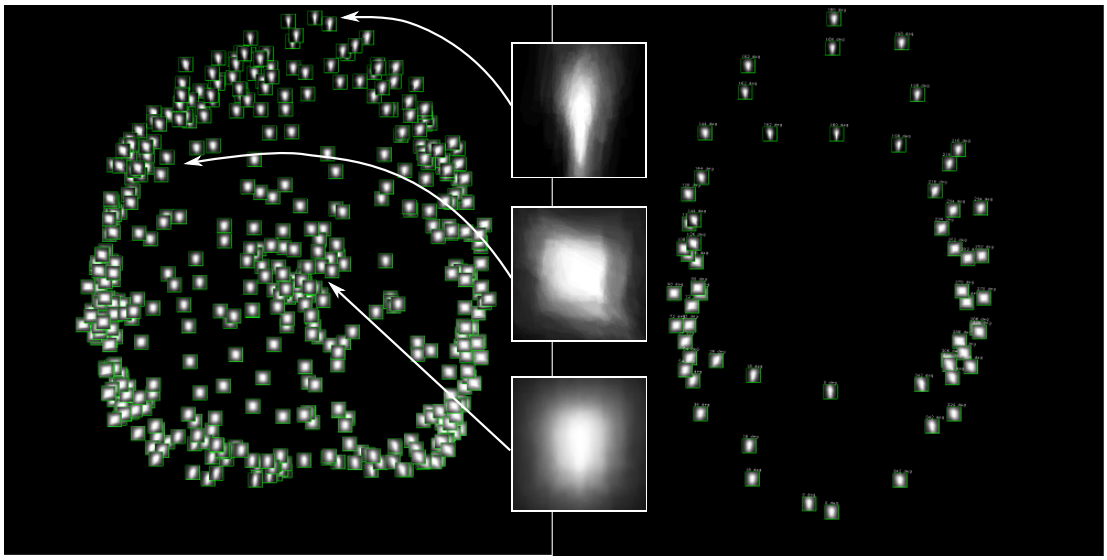


Figure 6.2: Scatter plot of tasks (left) and models (right), according to their first two factors. Models are arranged in a circular pattern according to the angle they were trained from, as are tasks. However, the center of the task circle is filled by tasks with very wide angular spreads, because they equally favor all models. Each "point" is the average silhouette of the positive videos for that task (best viewed under magnification in digital copy).

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.

This illustrative example is tested by constructing a scenario using synthetic (rendered motion capture) data. The goal is to recognize "walking" in video, and the only manipulated source of variation is the viewing angle. Each model in the library 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, $120^\circ - 155^\circ$); see Fig. 6.1. 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. 6.2), it can be seen that that the first two factors do in fact 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 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 ($0° - 360°$), there is no reason to prefer any model over another, corresponding to factors of (0,0), which rate all models equally.

For more complicated models and tasks, the "meaning" of the computed factors is rarely so obvious. Some factors for the synthetic data are visualized in Fig. 6.3. 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, that is, actions where there is no aggregate whole-body movement over the course of the action.

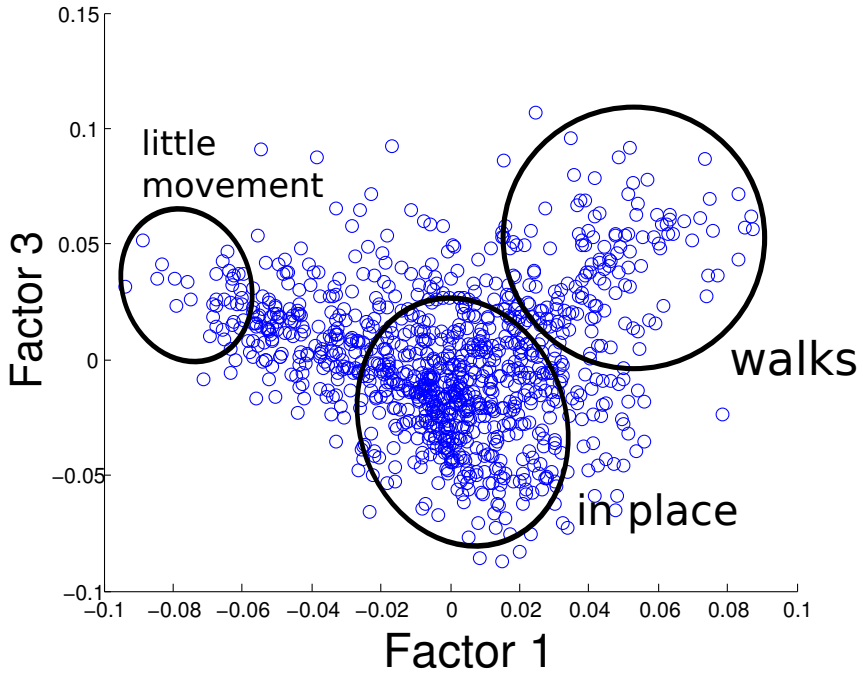


Figure 6.3: Visualized factors for tasks in the semi-synthetic actions dataset.

6.3.2 Factorization as Projection Onto a Basis

Instead of seeing each rating as being the product of latent factors vectors, instead all the ratings on a task can be seen as a linear combination of basis tasks (what might be called "eigentasks" in homage to the well-known Eigenfaces [101]). That is, instead of interpreting the matrix F as rows, where each row corresponds to the factors for a model, to interpret it as columns, where each column is an "eigentask".

As an illustrative example (see Fig. 6.4) for this interpretation, a more complicated synthetic situation is constructed with classifiers tuned to different viewing positions (angle from horizontal and distance), and tasks which vary in a similar manner. A library of 1600 viewpoint-tuned classifiers is trained (in a uniform 40×40 grid over angle from horizon and distance) and used to produce a ratings store of their accuracies on a number of synthetic tasks to detect walking.

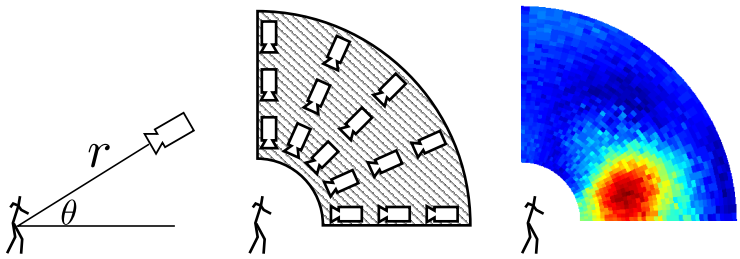
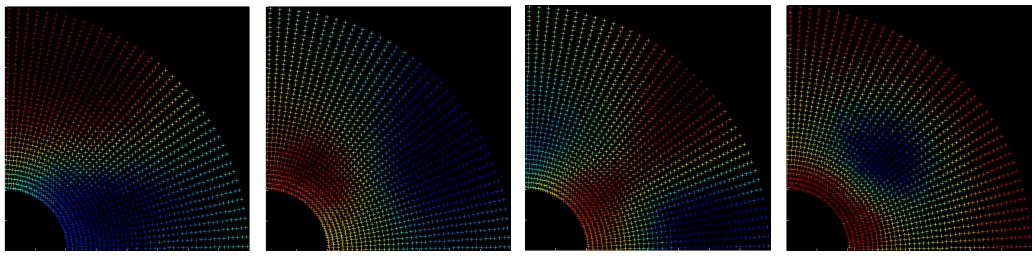


Figure 6.4: A library of classifiers produced by training SVMs to detect walking, where each SVM is trained only on samples from a narrow viewpoint, defined by its elevation θ and mean to subject r (left). The accuracies of all 1600 classifiers on a training set can be visualized as a heat map (right).



(a) 1st "Eigentask" (b) 2nd "Eigentask" (c) 3rd "Eigentask" (d) 4th "Eigentask"

Figure 6.5: The first four "eigentasks" from the factorization of the synthetic scenario in Fig. 6.4.

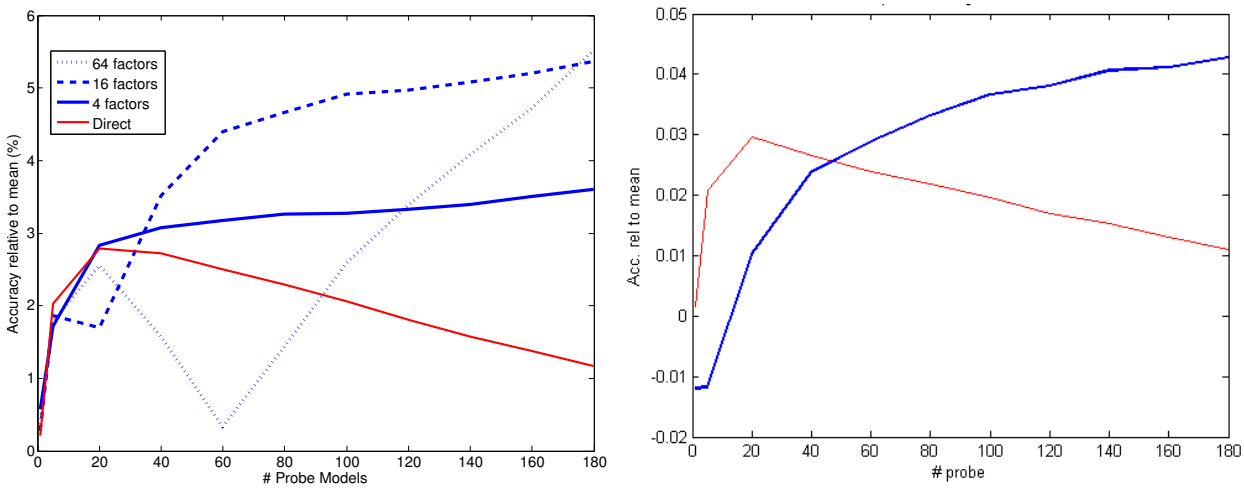
After the factorization, the F matrix can be interpreted as a basis for reconstructing the model accuracies on a task: see Fig. 6.5 for the first four columns of this basis. Note that these bases are essentially representing the spatial low-frequency variation between the walking detectors of the library. Thus, in this scenario the factorization method is literally performing a smoothing operation when it projects the known probe ratings onto this low-spatial-frequency basis and reconstructs all the model ratings.

6.4 Quantitative Results

The key question regarding single-model recommendation is whether the hypothesized ability to produce better recommendations than trying every model actually occurs in practice.

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; we call this method "direct selection", since it directly selects the model with the best apparent performance. When the number of probe models is equal to the total number of models in the library, direct selection corresponds to evaluating all the models in the library and selecting the one with the best measured performance. Since tasks vary in difficulty, to report performance across tasks we sometimes 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 library.

6.4.1 Mind's Eye Tasks



(a) Factorization (b) Sparse Coding

Figure 6.6: 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.

For the Mind's Eye dataset (see Sec. 3.1.4) tasks are generated as 1-vs.-1 action classification problems from the real Mind's Eye data, where each task has a (visible) training set of 2–16 labeled videos and a (hidden) test set of 6–40 (all the remaining videos for that 1-vs.-1 pairing).

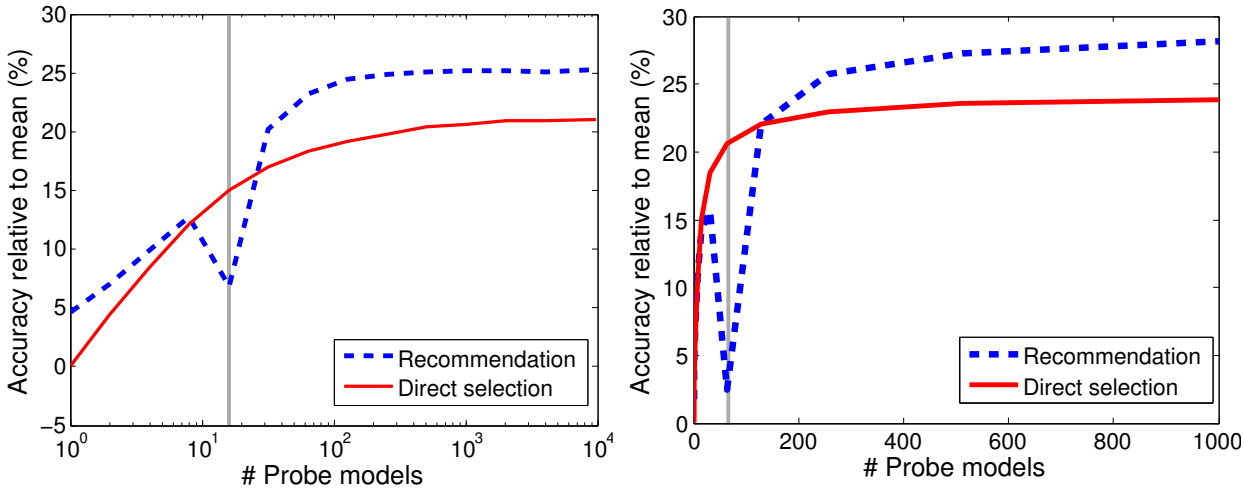
A library of 180 1-vs.-1 classifiers is built from semi-synthetic rendered motion capture data of the same four actions present in the Mind's Eye real data tasks, where each classifier is trained from 100 rendered video clips, 50 from each of two actions chosen at random for that classifier (model). Since there is so little real data, synthetic 1-vs.-1 tasks are also used to generate the ratings store from which the recommendations are made.

Both direct selection and recommendation see only the accuracies (ratings) of models as measured on the training sets of the tasks, but results are reported in terms of classifier accuracies on the hidden test sets. These results can be seen in Fig. 6.6. 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).

6.4.2 UCF50 and Semi-Synthetic Motion-Capture Tasks

UCF50 contains approximately 6600 videos, divided into 50 actions, with each action further subdivided into groups of videos, where each group comprises a set of (typically four) related



(a) Synthetic (b) UCF-YT

Figure 6.7: 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.

videos. Since all the videos in a group are closely related, the intention of the dataset is that videos from the same group should not be used to both train and test. However, for our purposes (limited data, multi-task evaluation), we take advantage of the groups by treating a small set of groups as a "task", so that each task is (by itself) relatively "easy", but is complicated by the limited training data available for the task as well as the variation between each task and the other tasks in the database.

First, all the groups are split into two partitions, where 2/3 of the groups are used to train classifiers for the library and generate the ratings store, and the other 1/3 of the groups are used to generate test tasks for evaluation.

We produce each test task by merging 1-3 groups of the same action, dedicating 2/3 of the videos in the merged set to the hidden test set, and the remaining 1/3 to visible training set for that task. This results in tasks with on the order of 2–10 positive training samples, and twice as many positive testing samples; we augment each task with an equal number of negative samples drawn at random from the other actions, so that each task is then a one vs. all binary classification problem with an equal number of positive and negative samples (so that chance is 50% accuracy). The mean number of training samples per task is 10.2, and the mean number of testing samples 24.3. Each group may be used in multiple tasks (but there is no overlap in terms of groups or videos between the tasks used to generate the ratings store and the test tasks used for evaluation).

The model library is generated by merging 3-10 groups of the same action for use as the positive training samples, and randomly sampling an equal number of negative training samples from the groups of the other actions. Keep in mind that there are only approximately 24 groups total available for each action in UCF50, of which 8 are reserved for testing, meaning that even a classifier trained on "all" the data for one action would only have 64 positive training examples.

With 3 to 10 groups per action, this means models in the library are trained from 12 to 40 positive examples (24 to 80 total). In this way 1000 models are generated (20 per action), since each model is trained on a different random selection of the available training data for an action.

The 2/3 partition of training groups are also used to generate generate a ratings store of the 1000 classifiers in the library rated on 1000 tasks. Thus, the ratings store has size 1000×1000. The store tasks are generated in the same way as the evaluation tasks, only they use the reserved training groups and not the test groups.

The models and tasks are generated in the same way for the rendered motion capture data, except that each group in the synthetic data is generated by choosing a random motion capture snippet and rendering it multiple times with randomizations and distortions. For the semi-synthetic data, since we can generate as much data as we want, we consider a library of 10,000 models rather than the 1000 generated for the UCF50 data.

As with the Mind's Eye data, direct selection and recommendation see only classifier accuracies (ratings) as measured on the training set of each task, but results are reported on the hidden test set. These results for the synthetic and UCF50 tasks are presented in Fig. 6.7a and Fig. 6.7b. 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.

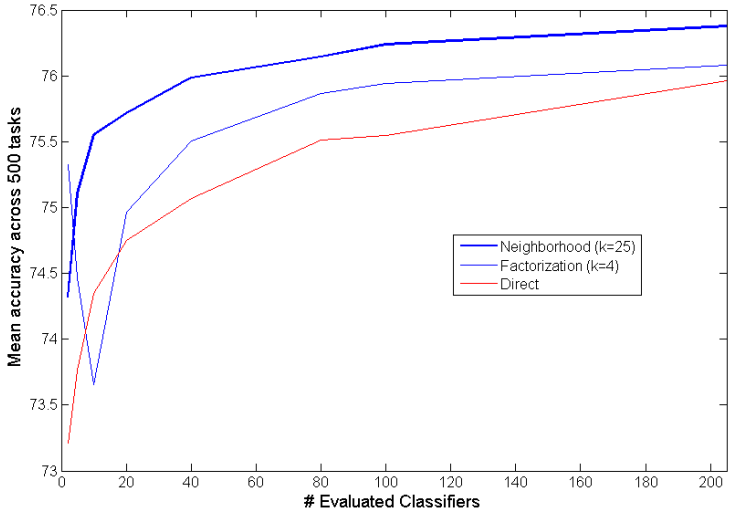


Figure 6.8: Mean relative accuracy vs. number of probe models for UCF50 dataset using ActionBank features.

Results for the UCF50 tasks with ActionBank features can be seen in Fig. 6.8. In this case, each "model" is not a trained classifier, but one of the entries in the action bank.

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 the number of probe models approaches the total number of models in the library. However, the disparity between model selection and direct selection can be seen as the cost of overfitting.

In addition, for the UCF50 tasks we also directly train SVMs on the STIP+HOG3D BOW histograms of the training set for each task; this method ("direct training") obtains an accuracy 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%.

However, it is hard to generalize from this result, since the comparative performance of choosing a model from the library versus directly training a model on the target task depends heavily on the application as well as the strength and diversity of the models in the library. In this case, direct training is limited primarily by the restricted amount of training data, while the strength of the model library is constrained by the limited amount of variation in its separate training data. The relative performance of the two depends on which one is limited more strongly. The model library is derived from training data which only really has at most 16 variations for each action (two thirds of the approximately 24 "groups" of videos for each action), and so with such limited variation, it is relatively unlikely that there will be an exact match to any test task in the library.

To get around this limitation of needing to have a very good match in the library due to the selection of only a single model, we later show how to combine *multiple* recommended classifiers to improve this result further (see the ensemble recommendation method in Chapter 7).

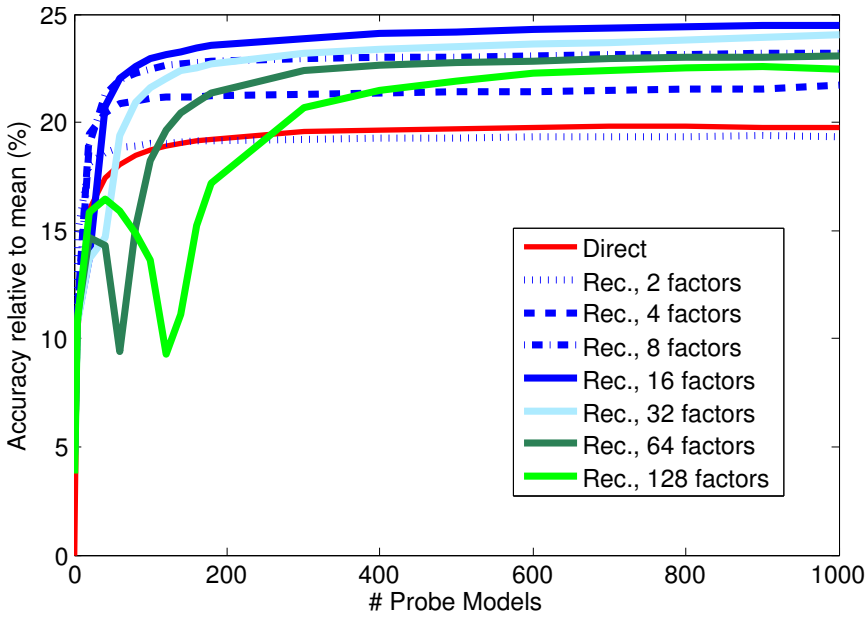


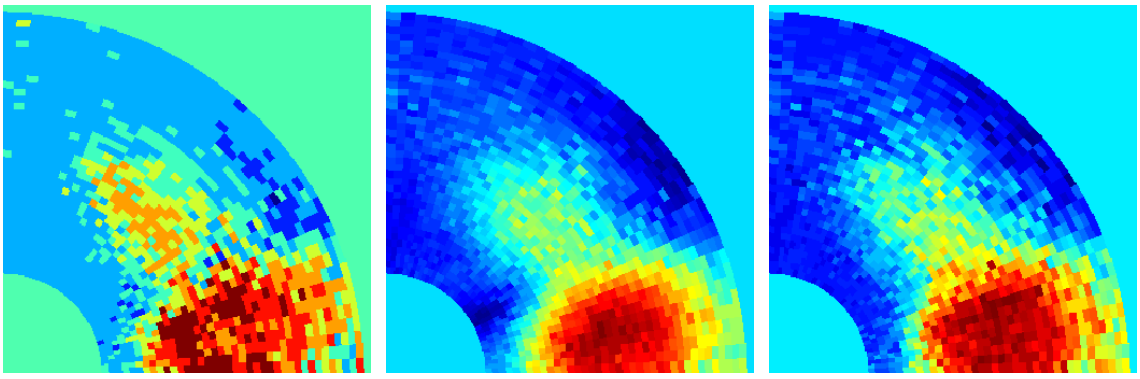
Figure 6.9: Effect of the number of factors in the factorization model for the synthetic tasks.

An interesting effect (Fig. 6.6) for the ME data and Fig. 6.9 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 evaluated 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 more than 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. 6.6b). In a later section (Sec. 6.6) we demonstrate how a regularized least squares solution can be used to fill in this cusp when using a factorization approach.

6.5 Search vs. Smoothing



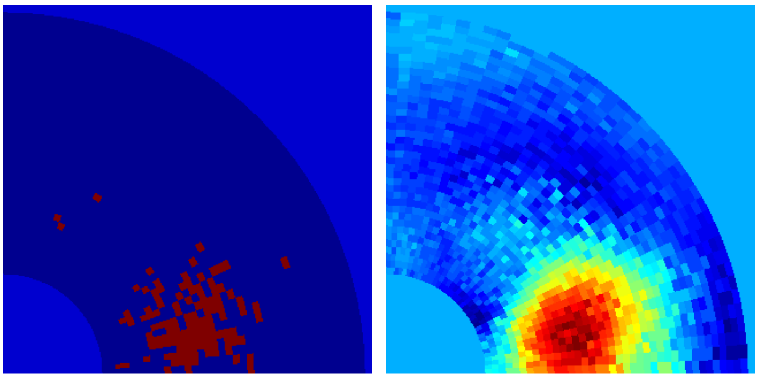
(a) Train set ratings (b) Predicted ratings (c) Test set ratings

Figure 6.10: 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. 6.4.

It has seemed as if there are two discrete modes of operation for model recommendation. The first is a smoothing mode, in which the measured ratings are assumed to be noisy and the model recommendation's benefit is primarily to *smooth* the noisy ratings into better estimates of the 'true' ratings. For example, the running example in which the goal is to pick a walking detector falls into this category, because the measured accuracies of the walking detectors on a (highly limited) training set are but noisy estimates of the true accuracies of those detectors. Fig. 6.10 illustrates this ability using the synthetic example of the viewpoint-tuned classifiers to demonstrate how recommendation is able to do better than directly rating every model; Fig. 6.11 illustrates a more extreme case where the measured accuracies (ratings) are completely binary.

In the second mode, the measured ratings are assumed accurate, and the purpose of the recommendation is to reduce the computation time by limiting the number of models that must be rated. In this mode, the recommendation serves to bias the order in which models are evaluated or searched to find the best performer. For an example, see Fig. 6.12 in which only twelve evaluated models are used to predict the ratings of the remainder of the models.

However, in fact both of these modes are merely extremes of a single search-smoothing continuum. The continuum is parametrized by how much to trust the measured ratings vs. the



(a) Raw Accuracies (b) Smoothed (Predicted) Ratings

Figure 6.11: An extreme example, in which the measured accuracies (left) are evaluated using only *one* training sample, resulting in a binary accuracy map in which each model's rating is either 0% or 100%. Despite this extreme quantization, model recommendation is able to smooth the ratings into a better estimate of the models' true accuracies (right).

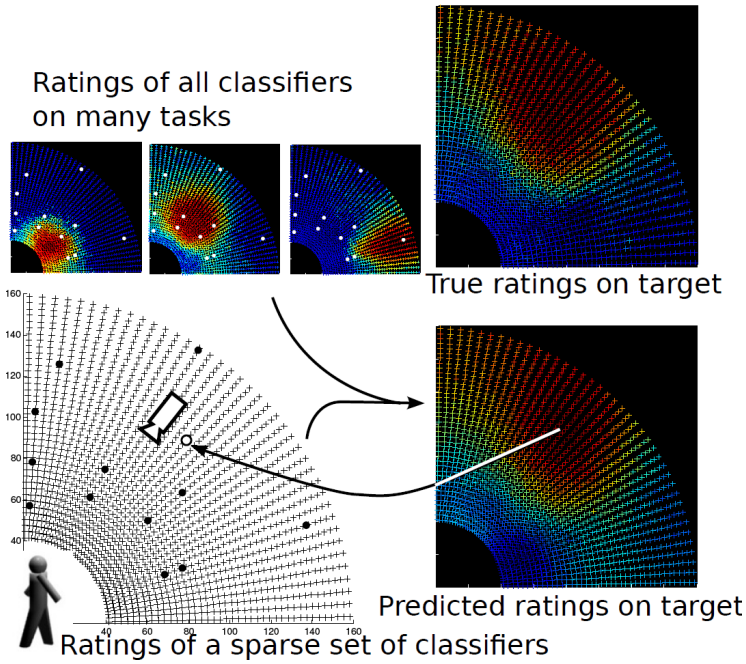


Figure 6.12: An example of model recommendation as "search": given twelve accurately evaluated model ratings, model recommendation is able to predict the remaining model ratings with startling accuracy.

predicted ratings; on one end, the trust is entirely in the predictions (smoothing) and on the other end, the trust is entirely in the measurements (search). Surprisingly, there is a middle region, where the correct behavior is not to trust either the predictions or the measurements entirely, but to take a weighted combination of the two.

6.5.1 Unified Search-Smoothing Algorithm

Algorithm 2 Unified Search/Smoothing Recommendation

Require: Total number of probes to evaluate, p_t
Require: Number of probes to search, p_s
Require: Blending weight between prediction and measurement, α
$P \leftarrow \text{SelectRandom}(L, p_t - p_s)$
$r_p \leftarrow \text{RateModels}(P)$
$r' \leftarrow \text{RecommendationPredictAllRatings}(P, r_p)$
$r'' \leftarrow \text{SortDescending}(r')$
for $i = 1, \ldots, p_s$ **do**
$r_i^* \leftarrow \alpha r_i'' + (1 - \alpha) \cdot \text{RateModel}(r_i'')$
end for
$s \leftarrow \text{argmax}_i \, r_i^*$
return A recommended entry from the library, s

Suppose that the hidden rating function (we shall be generous and suppose there actually is such a function) can be expressed as the sum of a linear part and a non-linear part. Then, the factorization method measures the linear part, with some error, while the direct measurement returns the linear plus non-linear parts, but with its own error. Then, the ratio at which the two should be blended to produce the best estimate of the true quantity depends on the ratio between the magnitude of the non-linear portions compared to the magnitude of the measurement error when directly rating.

Hence, the recommended model should not be chosen necessarily according to either its directly measured rating or its predicted rating, but by a linear combination of the two. Algorithm 2 summarizes this unified approach. In brief, first some number of models are evaluated as probes at random and used to predict (through the recommender system) the ratings of all the models. Then, an additional number of models are searched (rated/evaluated) in descending order of their predicted ratings. Finally, the model is returned which has the highest linear combination of its directly measured rating from the search step and its predicted rating from the recommendation. The blending factor for the linear combination is denoted α, and at $\alpha = 1$ the algorithm reduces to simple model recommendation (since it just picks the model with the highest predicted rating), and at $\alpha = 0$ it uses the recommendation system to prioritize the order in which models are directly rated on the test task, but in the end fully trusts the direct rating.

6.5.2 Quantitative Results

The 3D room layout application (Sec. 3.2) is a good example for search, since the correlation between the estimated and true ratings of models is relatively strong. In brief, there are on the order of 12,000 models (3D room meshes) and 216 room images (tasks), and the objective is to choose the best model matching a particular image. The "true" matching score is based on comparing the rendered geometric surface normals of the mesh (model) to the painstakingly hand-annotated normals in the image. The "estimated" matching score is based on comparing

the rendered mesh to the image based on various automatically extracted features, including automatically estimated normals (see Satkin *et al*. [90] for details; they have graciously provided the true and estimated ratings matrices for us to evaluate on). Each room is considered a task, and the ratings store is constructed by the ratings of the remaining rooms (leave one out cross validation).

Results for this application can be seen in Fig. 6.13. The unified search-smoothing algorithm (Algorithm 2) is used, with $\alpha = 0.3$ and the number of search probes $p_s = 200$ (that is, for a total number of probes p, evaluate $p - 200$ at random to do the recommendation, and then search the top 200 recommendations according to Algorithm 2).

Note that recommendation does improve on direct selection even when all the models are evaluated, but that the benefit is less dramatic than in previous applications because the correlation between estimated and true ratings is so strong. Indeed, in a hypothetical application where the estimated and true ratings were identical (so that the true ratings could be measured directly), it would not be logically possible to do better than trying everything and picking the best.

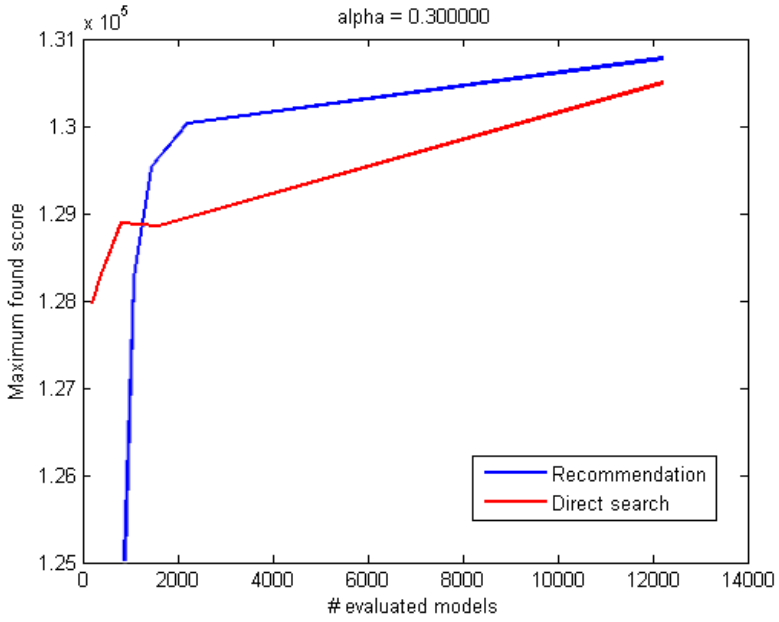


Figure 6.13: Match score of selected model vs. number of probe models for 3D room layout matching dataset, using the search algorithm with 200 searched models.

In the robotic state machine selection application, the objective is to select a good state machine to control a robot in a room coverage task for an unknown room layout (see Section 3.4 for details). There are 200 state machines in the library, and approximately 500 test tasks (room layouts); the recommendation is done from a ratings store with 1000 tasks. A rating is how much of the room the robot is able to cover in a time limit when controlled by a particular state machine. In this case, the difference between the estimated and true ratings is that there is random variation in the runs of the state machines, and so the estimated ratings are just one sample from the distributions of ratings (coverages) in which the means are the true ratings.

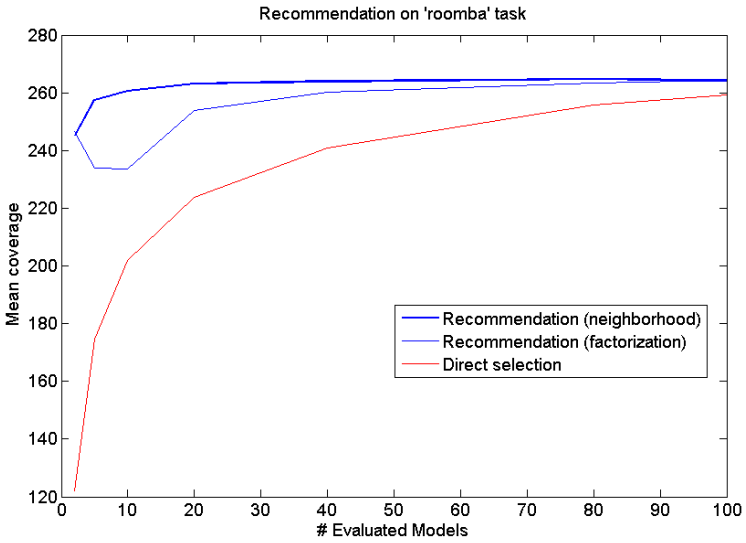


Figure 6.14: Mean coverage of the chosen state machine for the robotic state machine recommendation application, using both factorization ($k = 4$) and k-nearest-neighbor recommendation ($k = 20$), using the smoothing algorithm.

Fig. 6.14 gives results for recommendation in the robot state machine selection application. Note that this is a case where the simple k-nearest-neighbor recommendation method gives better results than factorization. This is more likely when there are only a few models to choose from, because the factorization method tends to more aggressively smooth over variations in model performance.

In the ego-centric skin detection application, the goal is to detect skin (per-pixel skin classification in a video frame). Each model is a trained skin detector (per-pixel classifier), and each task is a different frame of video. In this case, the models and tasks are all for the same person, but vary in terms of the scene illumination. Thus, the objective is to pick the best illumination-tuned skin detector for a particular video frame. This data was provided by Cheng Li and Kris Kitani, and is described in more detail in Sec. 3.3.

In brief, there are 100 trained skin-detectors (per-pixel skin classifiers), and for a test task (a video frame) the objective is to pick the one with the highest F1 score (harmonic mean of precision and recall) for that frame. The detectors' training data and test tasks are drawn from the same long (18000 frames) video sequence, with the training taking place on the first 1000 frames and testing on the remainder. Since ground-truth annotations are only provided every 50 frames for the testing portion, this results in 340 test tasks (frames).

Fig. 6.15 shows results on the ego-centric skin detection dataset, using ratings which are the ground-truth F1 scores of the detectors, but with varying amounts of noise added. This emulates a situation in which there is some way of estimating the quality of the detection without having the underlying ground truth, albeit with some error. Estimating the quality of algorithms without ground truth has been explored by Aodha *et al*. [62] and Jammalamadaka *et al*. [40], so this type of estimation is within the realm of possibility.

In the case where zero noise has been added, the problem is a pure search problem, and trying

all the models and selecting the highest rated one is the optimal thing to do. In that case, model recommendation does slightly worse when all models are searched, but can otherwise significantly reduce the number of models that must be searched, provided some loss of performance in the selected model is allowed. However, with increasing noise, the recommendation does better than direct selection even when all models are tried, since the model recommendation is able to smooth over the added noise in the ratings.

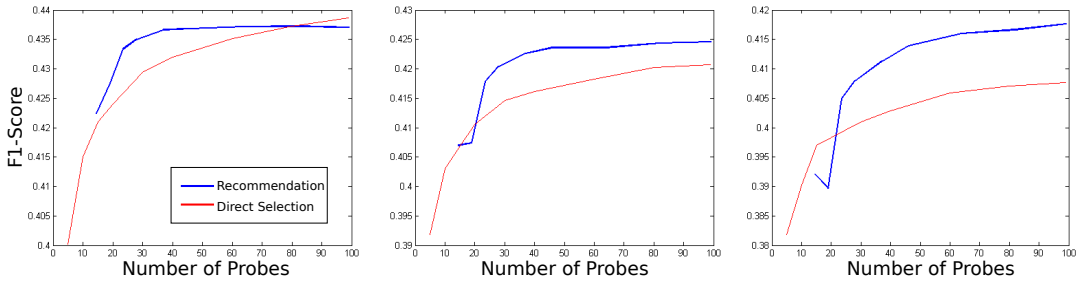


Figure 6.15: Results for the egocentric skin detection application, where increasing amounts of noise have been added to the ratings. Model recommendation in blue, direct selection in red. Left: zero noise has been added (ratings are perfect). Center: Normally distributed noise with $\sigma = 0.05$ has been added. Right: Normally distributed noise with $\sigma = 0.10$ has been added.

6.5.3 Effect of the α Parameter

In the previous section we presented results using the pure search algorithm ($\alpha = 0$), but in this section we perform experiments to vary the alpha in different applications to see whether selecting by a blend of the estimated and predicted ratings can improve performance.

Fig. 6.16, Fig. 6.17, and Fig. 6.18 show plots of the estimated vs. ideal ratings for the model libraries on the 3D room layout, UCF50, and synthetic action datasets, respectively, as well as the accuracy obtained by varying the α parameter that controls the weighting between estimated and predicted ratings for selecting the best model. The left subfigure in each case plots the estimated and true ratings against each other as a scatter plot; if the true ratings could be exactly measured, the plot would be a perfect diagonal line. Since the true ratings cannot be directly measured, the scatter plots instead form clouds of uncertainty. The right subfigure in each case plots the mean accuracy across tasks (when every model is used as a probe) against the α parameter that blends the predicted and estimated ratings for recommending a model.

Note that in applications where there is less noise (3D room layout), the best accuracy is achieved by setting α to prefer the estimated ratings rather than the predictions ($\alpha \approx 0$), whereas in datasets where there is more noise in the estimates of the ratings, the best accuracy is to set α to trust the predictions ($\alpha \approx 1$).

An interesting phenomenon that can be seen in the scatter plots of estimated vs. ideal ratings is the effect of quantization in the estimated ratings. In the applications which are classification based, when measuring or estimating accuracies from small training data, the estimated accuracies are quantized according to how many training samples are available (*e.g.*, if there are ten training samples, then accuracies can only be measured in multiples of $\frac{1}{10} = 0.1$). This can be

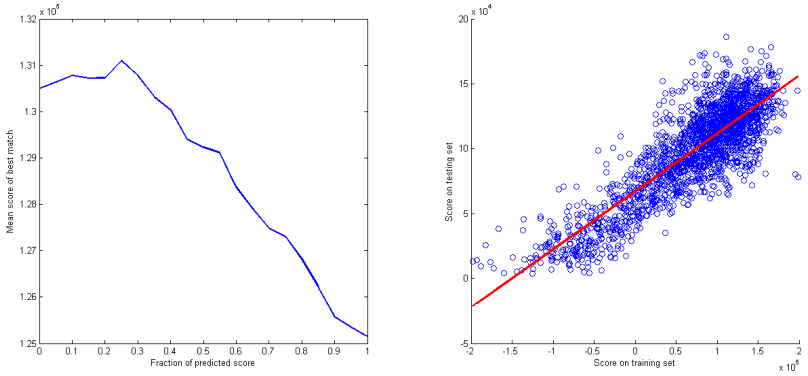


Figure 6.16: Estimated vs. Ideal ratings for the 3D room layout application. Left: plot of measured ratings vs. true ratings for each model in the library. Right: effect of varying the relative weighting between measured ratings and predicted ratings in order to select the "best" model.

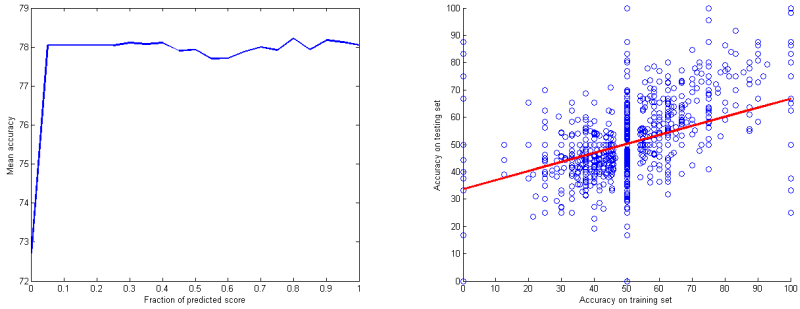


Figure 6.17: Estimated vs. Ideal ratings for the UCF50 dataset. Left: plot of measured ratings vs. true ratings for each model in the library. Right: effect of varying the relative weighting between measured ratings and predicted ratings in order to select the "best" model.

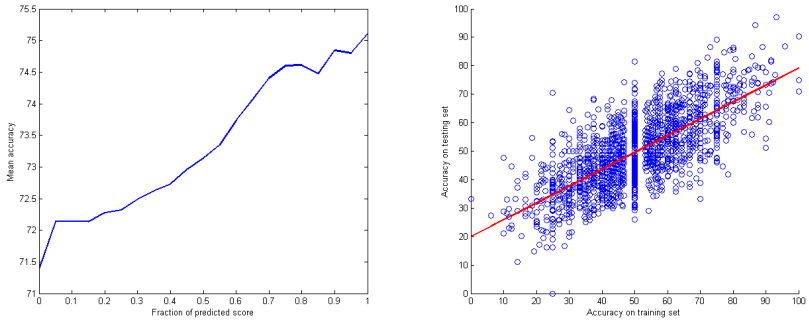
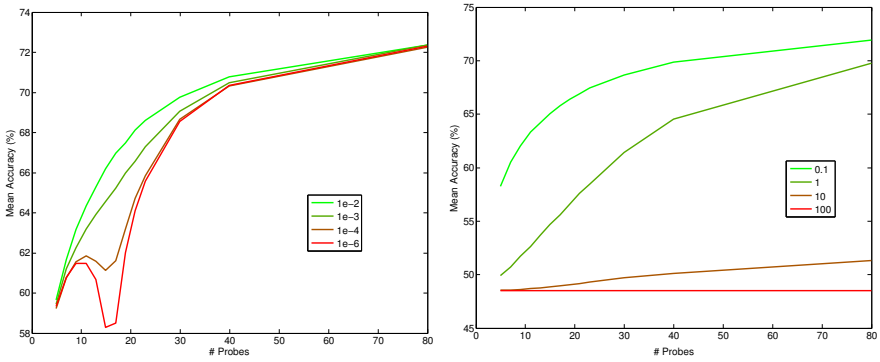


Figure 6.18: Estimated vs. Ideal ratings for the synthetic action dataset. Left: plot of measured ratings vs. true ratings for each model in the library. Right: effect of varying the relative weighting between measured ratings and predicted ratings in order to select the "best" model.

seen as a two prominent horizontal bands empty of points in Fig. 6.17 and to a lesser extent in Fig. 6.18. The reason that these bands only appears once is that for each task the quantization 'step' is slightly different, but they all create an excluded zone around zero.

6.6 Regularized Least Squares and the "Cusp"



(a) Low Regularization (b) High Regularization

Figure 6.19: Mean accuracy vs. number of probe models for the semi-synthetic dataset, using 16 factors with the factorization approach and the regularized least squares solution, under different regularization parameters λ.

In the factorization method, the k factors for a new task are found by solving a least-squares problem. The method in which the least squares problem is solved is largely unimportant unless the number of evaluated probe models (the number of known probe ratings) is close to or less than the number of hidden factors k. If $p < k$ then the problem is underconstrained, and to be solved some manner of regularization must be used; if the pseudo-inverse solution is chosen, then the implied regularization is to penalize the norm of the factors vector x. However, if $p = k$ then $\hat{F}^T$ is "almost certainly" invertible, yielding a unique solution. Counter-intuitively, this means that, when using the pseudo-inverse, the result may be worse when $p = k$ than when $p < k$, because there is no regularization in the $p = k$ scenario to force the factors x to take reasonable values. Indeed, since the factorization predicts the *residual* ratings, a regularization term penalizing the norm of x is consistent with the intuition that the residuals should be small.

In practice, this means that, when using the pseudo-inverse or equivalent linear solution, there may be a "cusp" or dip in performance around the point where $p = k$, as can be seen in Fig. 6.7a. If performance in this low-probe-count region is important, then instead of finding the standard least-squares solution which minimizes $||\hat{F}^T x - \bar{r}_p||_2$, a regularized least squares solution can be found which minimizes $||\hat{F}^T x - \bar{r}_p||_2^2 + \lambda||x||_2^2$, where λ is a factor which controls how much to penalize the norm of x. This regularized solution is solved by solving the augmented least squares problem

$$\left[\frac{\hat{F}^T}{\sqrt{\lambda} I_{k\times k}}\right] x = \left[\frac{\bar{r}_p}{0_{k\times 1}}\right], \tag{6.6}$$

where x is the vector of hidden factors being solved for.

Figure 6.19a demonstrates the effects of this regularization on the semi-synthetic dataset (here restricted to 1000 models to focus on the effects in the cusp region) when the regularization parameter is low. Note how the regularization is able to remove the cusp without being detrimental to performance with more probes. However, if the regularization parameter λ is set too high, then the regularization takes a toll on accuracy, as can be seen in Figure 6.19b.

Chapter 7

Ensemble Recommendation

The core model recommendation framework and the collaborative filtering literature concentrate on recommending a *single* model or item from the library. There has been little work on recommending sets in collaborative filtering, likely because there are no datasets or applications where there are sensible reasons to jointly recommend 'baskets' of items. Although Amazon.com or Netflix may give users lists of recommended items, these items are all recommended independently, as the systems do not try to recommend movies that should be watched together or books that should be read in sequence for the most enjoyment. That is to say, for consumer items, there is usually not any natural way of *combining* items that would be different from simply consuming them independently.

In contrast, in many model recommendation applications it is natural to want to combine models, for example, in classification problems where the idea of combining weak classifiers together into stronger ones is common (for example, in boosting [36]). From a recommendation standpoint, this creates a problem, since the performance of a combined set or *ensemble* of classifiers is not a function of their individual performances. An extreme example is that a set of the same classifier twice gives no performance benefit over the same classifier once; this is different from, say, muffins, where eating two muffins is (for many people!) certainly better than eating just one.

This extreme case cuts to the heart of the problem with ensemble recommendation, namely, that unless treated with care, the recommendation system could recommend a set of models which are so highly redundant as to offer little or no benefit over the individual models.

In machine learning, for the purpose of choosing an ensemble of classifiers these types of classifier interactions are usually resolved by iterative greedy methods, such as boosting [36], where at each iteration another classifier is added to the set which minimizes some notion of error.

Given the popularity of boosting, it has unsurprisingly been applied to multi-task learning as well. These methods tend to follow the standard multi-task approach of enforcing sparsity, such as in Chapelle *et al*. [21] where boosting selects a common set of weights for weak learners across all tasks, and then individual tasks are allowed to sparsely deviate from that common weighting. Wang *et al*. [103] take a slightly different approach, where the sparsity is enforced by learning a partitioning (clustering) of the tasks, where all the tasks in a cluster are forced to share the same weights for the weak learners. Faddoul *et al*. [34] take yet another approach, in

which the weak learners are joint classifiers of two tasks, and so the boosting naturally selects a compromise between two tasks. However, the limitation of their approach is that it does not easily scale to more than two tasks, and these techniques generally need closely related tasks.

7.1 Ensemble Recommendation Methods

We consider four options for selecting classifier ensembles. The simplest, top-k recommendation, just selects the top-k classifiers according to their predicted accuracies from model recommendation. Set expansion expands the library to contain not individual models, but rather sets of models, reducing the problem of recommending a set back to the original problem of recommending a single library item. AdaBoost is an unmodified, standard boosting algorithm. Recommendation boosting takes AdaBoost, but uses model recommendation to select the classifier at each iteration. Recommendation boosting+ uses the same underlying mechanism as recommendation boosting, but afterwards combines its selection with the top-k selection to add more variation to the selected set.

Note that we use the boosting methods as feature selection mechanisms and discard the final weights of the selected classifiers in favor of simply training an SVM on the selection; this use of AdaBoost is common in vision [94, 112] and occasionally sees use in other domains [74, 113].

7.1.1 Top-k Recommendation

Given the predicted ratings r' according to recommendation, rather than selecting only the top one, we select the top k, where k is the size of the desired set to be recommended. This is the obvious way of recommending multiple classifiers, but the downside is that it can potentially recommend a highly redundant set since it does not consider interactions or synergies between recommended classifiers.

7.1.2 Set Expansion

Supposing that the library contains b base classifiers, then a classifier set j is produced by selecting a set of these base models, represented by the selection vector s_j, where an element $s_{jb} = 1$ if the classifier set j includes the base model b. Note that in practice a classifier set functions as a single classifier by training an SVM (or other classifier) on the outputs (decision values) of the base models it is built on.

Set expansion populates the library with sets of classifiers rather than the individual classifiers themselves. To compare set expansion to top-k on even ground, we use the same library size for both (1000 in the quantitative examples). In general, these sets might be generated in many ways, but in order to make use of all the original classifiers we generate the sets in the following manner: first, start with a library of all empty sets (1000 in our case). Then, for each original classifier, randomly assign it to k sets in the library. Thus, each original classifier is guaranteed to be present in k sets. If there are m sets in the library and n classifiers in the original library, this procedure results in sets with a mean size of $\frac{kn}{m}$, in our case 20.

For the purposes of recommendation these classifier sets can be treated identically to base models, requiring no change in the underlying algorithm.

7.1.3 AdaBoost

Although the AdaBoost has been extended to more than two classes, for simplicity we consider only the binary classification problem. The algorithm learns a classifier from a training set, where X_i is the ith data sample in the training set of a target task, and $y_i \in \{-1, 1\}$ is the associated binary label for that sample, and where n is the number of training samples.

AdaBoost is an iterative algorithm, where each iteration considers a different weighted version of the training set; we denote the weight of data sample i in iteration t by w_{it}. Then, given a classifier f_j, the weighted error of that classifier at an iteration is given by

$$\text{WeightedErr}(f_j, W, X, y) = \frac{\sum_i I(f_j(X_i) \neq y_i) \cdot w_{it}}{\sum_i w_{it}}, \tag{7.1}$$

where $I(.)$ is an indicator function. The weighted accuracy of the classifier is given by the straightforward expression $\text{WeightedAccuracy}(f_j, W, X, y) = 1 - \text{WeightedErr}(.)$.

At each iteration the classifier with the lowest weighted error is selected, and the weights modified to increase the weights of misclassified samples and decrease the weights of correctly classified ones. The algorithm is given in Alg. 3.

Algorithm 3 AdaBoost

Require: Classifiers $F = \{f_1, f_2, \ldots, f_j, \ldots\}$
Require: Training-samples and labels X, y
  $\forall w_i \in W, w_i \leftarrow \frac{1}{n}$
  $S \leftarrow \{\}$
  for $t = 1, \ldots, k$ **do**
    $s_t \leftarrow \text{argmin}_{f_j} \text{WeightedError}(f_j, W, X, y)$
    $e_t \leftarrow \text{WeightedError}(s_t, W, X, y)$
    $\alpha_t \leftarrow \frac{1}{2} \log \frac{1-e_t}{\max(e_t, \epsilon)}$
    $\forall w_i \in W, w_i \leftarrow w_i \cdot e^{-\alpha_t \cdot \text{sign}(y_i \cdot s_t(X_i))}$
    $W \leftarrow \text{Normalize}(W)$
    $S \leftarrow S \cup \{s_t\}$
  end for
    return A selected set $S \subseteq F$

7.1.4 Recommendation Boosting

We modify AdaBoost to incorporate recommendation using the the key insight that boosting algorithms can be seen as a series of tasks, and therefore model recommendation can be used to pick the weak learner at each iteration of the algorithm, rather than the error-prone direct evaluation typically used.

Algorithm 4 Recommendation Boosting

Require: Classifiers $F = \{f_1, f_2, \ldots, f_j, \ldots\}$
Require: Training-samples and labels X, y
Require: Ratings matrix R of classifier accuracies on other tasks
  $\forall w_i \in W, w_i \leftarrow \frac{1}{n}$
  $S \leftarrow \{\}$
  for $t = 1, \ldots, k$ **do**
    $A = [\text{WeightedAccuracy}(f_j, W, X, y) \text{ for } f_j \in F]$
    $s_t \leftarrow \text{argmax}_{f_j} \text{RecommendationPrediction}(f_j, R, A)$
    $e_t \leftarrow \text{WeightedError}(s_t, W, X, y)$
    $\alpha_t \leftarrow \frac{1}{2} \log \frac{1-e_t}{\max(e_t, \epsilon)}$
    $\forall w_i \in W, w_i \leftarrow w_i \cdot e^{-\alpha_t \cdot \text{sign}(y_i \cdot s_t(X_i))}$
    $W \leftarrow \text{Normalize}(W)$
    $S \leftarrow S \cup \{s_t\}$
  end for
    return A selected set $S \subseteq F$

Recommendation boosting simply replaces the selection of the classifier with the lowest weighted error with a model recommendation step. That is, instead of using the weighted errors of the classifiers to directly select the classifier for an iteration, the measured accuracies (along with a ratings matrix R of the accuracies of classifiers in the library evaluated on other action recognition tasks) are fed into model recommendation to predict the accuracies for the classifiers, and the classifier with the highest predicted accuracy selected. This modified algorithm is given in Alg. 4.

7.1.5 Recommendation Boosting+

As explained in [12], eventually AdaBoost will converge to a 'limit cycle' in which the same weak learners are cyclically selected. If the number of training samples is small, this convergence can happen very quickly. As a result, in the quantitative experiments performed later, AdaBoost only selects a mean of 11 unique classifiers over 20 iterations, while recommendation boosting only selects 10 unique classifiers over those same 20 iterations.

In recommendation boosting+ we add variety to the set of classifiers selected by recommendation boosting– if recommendation boosting only selects b unique classifiers, but a set of k is desired, then the remaining $k - b$ classifiers are selected as the $k - b$ classifiers with the highest predicted accuracies according to model recommendation.

7.2 Qualitative Demonstration

It is difficult to visualize the iterative selection, weight modification, and recommendation in the high-dimensional, unordered space of making recommendations for generic action recognition.

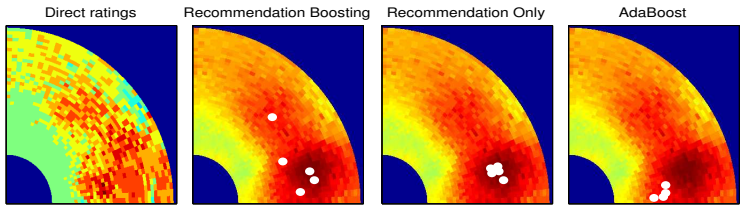


Figure 7.1: A comparison of the selected classifiers according to top-k recommendation, AdaBoost, and recommendation boosting. Top-k recommendation selects a highly redundant set of classifiers, while AdaBoost is led astray by a few erroneously good classifiers. Recommendation boosting selects a set of classifiers with better coverage of potential viewing conditions.

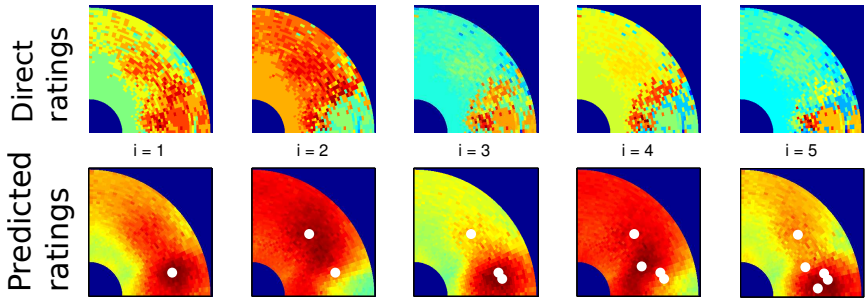


Figure 7.2: Progression of the classifiers selected by recommendation boosting; note how each iteration's re-weighting of the training samples shifts the distribution of predicted accuracies so that the selected classifiers do not all clump near one location, as in top-k recommendation (see Fig. 7.1).

Recommendation boosting can be more easily understood on the simplified example introduced in Fig. 6.4.

Qualitative results for the viewing angle situation can be seen in Fig. 7.1. Note how top-k recommendation chooses a very redundant set of classifiers, where all five classifiers are tightly clustered around the predicted maximum. AdaBoost, on the other hand, is confused by the spurious classifiers that appear to have 100% accuracy on the training set. Recommendation boosting picks a few classifiers near the predicted maximum, but then spreads the remainder out for better coverage of the region near the maximum. Thus, compared to the tightly clustered set of classifiers chosen by top-k recommendation, the recommendation boosted set is more likely to be robust to variations in the action. Note that since recommendation boosting selects five distinct classifiers over the five iterations, its selection is identical to recommendation boosting+; the two only differ when recommendation boosting selects non-unique classifiers.

A visualization of how recommendation boosting selects its classifier each iteration can be seen in Fig. 7.2. At the first iteration, the selected classifier is the same as the top recommended classifier, but then in subsequent iterations, as misclassified samples are more strongly weighted, the distribution of classifier ratings changes to promote the selection of classifiers other than those at the original maximum. At each iteration the method is able to smooth over the extremely noisy measured accuracies for the iteration to produce a better estimate of where the maximum accuracy is obtained for that iteration.

7.3 Quantitative Evaluation

Recommendation boosting can be directly compared to the single-model selection recommendation method presented in Chapter 6. As a reminder, this is an action recognition problem on the UCF50 dataset [102] using limited training data (10.2 training samples on average, compared to the approximately 100 per action that are available when UCF is evaluated as a single task). As before, models are SVM classifiers on STIP [52] plus HOG3D [45] bag-of-words histograms; this is commonly used as the foundation for action recognition systems and often performs similarly to more complex approaches [49]. The same library of 1000 models is used in this section as in the previous ones.

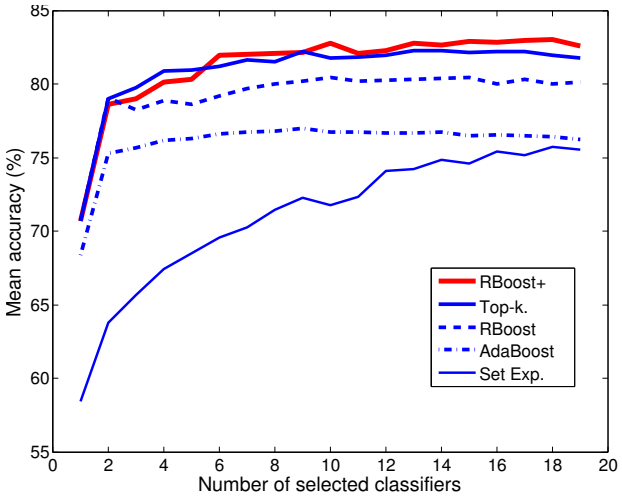


Figure 7.3: Mean accuracy of the selected set of classifiers vs. the size of the selected set. Top-k recommendation and recommendation boosting+ have the best performance, with recommendation boosting+ having a slight edge at larger set sizes. The accuracy obtained by directly training each task on the low-level STIP+HOG inputs is 77%.

Quantitative results on the UCF50 dataset are shown in Fig. 7.3, where it can be seen that recommendation boosting outperforms both AdaBoost and direct training. Interestingly, the straightforward top-k recommendation method does better than basic recommendation boosting, suggesting that for this evaluation domain, redundancy in the classifier library is not as large a concern as expected.

The classifiers used for the qualitative example were simply parametrized by just r and θ, and so it was easy to find redundant classifiers (because the $[r = 10, \theta = 0.3]$ classifier is very similar to the $[r = 11, \theta = 0.3]$ classifier). In the more complex action recognition problem, it is likely that the 1000 classifier library samples the space of possible classifiers so sparsely that there is too little redundancy in the classifiers to be detrimental. Sec. 7.4 explores the degree of redundancy in these datasets in more detail.

For this experiment, the direct training baseline (in each task, directly train an SVM on the input STIP+HOG bag-of-words histograms, rather than appealing to the library) obtains an accuracy of 77%, which only AdaBoost and set expansion fail to exceed. Recommendation boosting+

exhibits a slight improvement over top-k selection (this difference is statistically significant to $p < 0.05$ for set sizes ≥ 18).

All of the selection strategies except set expansion show large gains at first and then quickly plateau, indicating that they front-load their selections with the strongest classifiers. Indeed, the difference between the recommendation variants largely manifests after the first two selected classifiers.

Set expansion, on the other hand, starts very poorly but shows consistent gains. This is because set expansion does not order the classifiers in the sets in any way, so stronger classifiers are just as likely to appear later as earlier. The overall low performance of set expansion suggests that most of the classifiers are low performance, and thus the randomly generated expanded sets tend to contain mostly poor classifiers. For example, even if 10% of the classifiers are good for a particular task (probably an overestimate), the chance of generating a set of 20 with even five good classifiers is still vanishingly small.

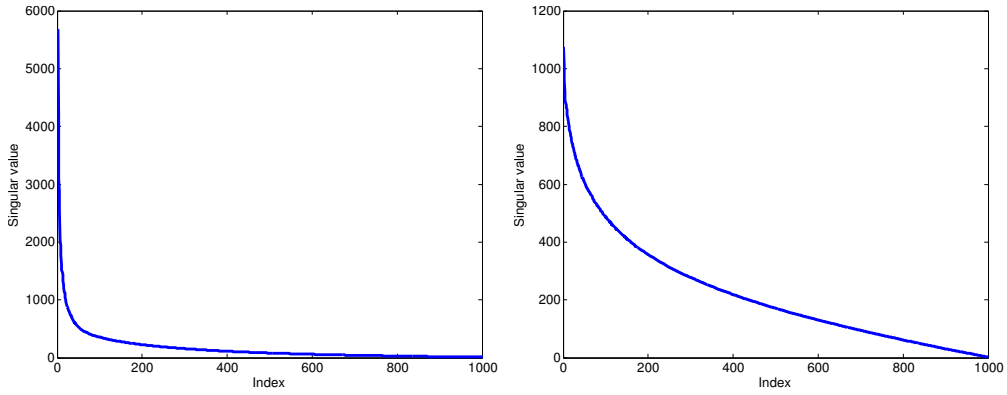


Figure 7.4: Singular values plotted by rank (ordering) for factorization of individual models and set-expanded models. Since much more of the variation is represented by the first factors for the individual models, the factorization method better represents the ratings of individual models than of sets. This indicates that the patterns in the sets' ratings are far less linear than for the individual models that compose them.

7.4 Redundancy

7.4.1 Defining Redundancy

It is important to distinguish between two types of redundancy: redundancy in the models themselves (which we term *mechanical* redundancy), and redundancy in the ratings of those models (which we term *correlation* redundancy). Note that while mechanical redundancy redundancy between two models implies correlation redundancy, the inverse is not true: two models may have identical ratings, yet have different underlying (and possibly complementary) implementations.

More specifically, two models m_1 and m_2 are mechanically redundant if and only if the ensemble $\{m_1, m_2\}$ receives substantially the same rating as both m_1 and m_2 on all tasks. That is to say, combining two mechanically redundant classifiers should not provide any benefit on any task.

Measuring mechanical redundancy is impractical, however, since it involves building n^2m ensemble models for n base models and m tasks; for 1000 base models by 1000 tasks, that is a billion models that must be produced, and if the models are classifiers that must be trained, this is a substantial computational cost.

Instead, we can measure correlation redundancy between models, which is to say measure how strongly the ratings correlate between different models. This will overestimate the redundancy in a model library, since there may be models whose ratings are strongly correlated but which can still be combined into an ensemble that outperforms both base models.

7.4.2 Artificially Increasing Redundancy

In the previous experiment we concluded that the models trained for the UCF50 dataset do not have enough redundancy between them that selecting sets by the direct top-k method would pose a problem. However, for some applications it may be the case that there are in fact highly redundant features, and so here we perform an experiment where we artificially increase the redundancy in the UCF50 model library to test the performance of the ensemble selection methods.

The way we artificially increase redundancy is simply by duplicating each model in the library a number of times, referred to as the *redundancy factor*. If each model appears only once, that is a redundancy factor of 1.0, whereas if each model has one duplicate added (so there are two copies of each model), that is a redundancy factor of 2.0. Since we are literally duplicating the models, we do not actually copy the classifiers, but instead duplicate the rows of the ratings matrix corresponding to those models, since the duplicate classifiers are deterministic and will produce identical ratings on identical tasks. To avoid numerical issues, we add a small amount of uniformly distributed noise to the ratings and classifier decision values.

Increasing redundancy in this way will not necessarily reflect the natural patterns of redundancy produced in real applications, but it provides a good model for a *worst case* redundant situation, where the models are truly mechanically redundant by being near clones of one another.

Method / redundancy factor	1	2	3	4
Top-k	20	10	7	5
Recommendation boosting	14	13	15	14
Recommendation boosting+	17	14	15	14

Table 7.1: Number of unique models selected by each method vs. redundancy factor

In Table. 7.1 we examine the number of unique (non-redundant) classifiers chosen by each method vs. the redundancy factor. Note that (as intuition suggests) top-k chooses $\frac{k}{f}$ unique models, where k is the requested set size, and f is the redundancy factor. That is, with a redundancy

factor of 4, top-k will choose only five unique models, because although it is trying to select 20 models, in fact it chooses the top five models four times each due to the 4-fold duplication.

Note that in this table, with a redundancy factor of 1.0 (no duplicates added), recommendation boosting chooses 14 unique models, whereas earlier we claimed that recommendation boosting only choose an average of 10 unique models. The difference is due to the fact that in this experiment the small amount of added noise has improved results for recommendation boosting, likely because it serves to break "ties" between models that would otherwise cause cycles in the no-noise-added situation. Recommendation boosting plus does not manage to select a full 20 unique models because the top-k list and the recommendation boosting list overlap at the top, on average by three models.

Nevertheless, recommendation boosting and recommendation boosting plus are much more resilient to the increased redundancy under these conditions. In fact, recommendation boosting is almost completely insensitive to the redundancy factor, since the number of uniques it chooses is determined by how large the limiting cycles (see [12]) of the boosting process are, and not by the presence of duplicates. Recommendation boosting plus shows gains over recommendation boosting for small redundancy factors, but since it takes its additional choices according to top-k, as the redundancy factor increases recommendation boosting plus will converge to the same selection as plain recommendation boosting.

7.4.3 Measuring the Redundancy of Real Applications

In this section we measure the correlation redundancy of model libraries produced for several applications.

Let c_{ij} be the correlation coefficient between models m_i and m_j as measured on their ratings across all tasks in an application. Then, we define the redundancy count C_R for that model with threshold β as

$$\mathrm{C_R}(m_i, \beta) = \sum_j I(c_{ij} \geq \beta). \tag{7.2}$$

Then, we can measure the empirical redundancy factor F_R of an entire model library in an application as the mean redundancy count, or

$$\mathrm{F_R} = \frac{\sum_i \mathrm{C_R}(m_i, \beta)}{n}, \tag{7.3}$$

where n is the number of models in the library, and β is again the threshold correlation for counting redundancy.

Tab. 7.2 shows computed redundancy factors (for $\beta = 0.9$) for the model libraries of several applications. Note that the robotic vacuuming application (see Sec. 3.4) and the 3D model matching applications do not currently have extensions to ensembles, and the redundancy results are presented only to demonstrate what levels of redundancy are present in different domains.

These results suggest that there is a surprisingly wide spread of redundancies: for the UCF50 classifiers, there is almost no redundancy whatsoever, but for the 3D model matching application, each model has on average 167 'clones'! The histogram of duplicate counts for the 3D model application shows that the distribution is bimodal: most models have relatively few duplicates,

Dataset	Estimated Redundancy Factor F_R
UCF50	1.08
Vacuum Policy (2D Layouts)	2.09
3D Room Layout	167.0

Table 7.2: Estimated redundancy factors for real datasets

but there are a large number of models with an extreme number of duplicates. This is because there are some 3D models which are never matched to test models, and so receive universally bad matching scores; these models are all considered duplicates of one another according to the correlation in scores.

One aspect this analysis does not account for is the fact that duplication in poorly performing models does not matter as much as duplication in well performing ones: that is, it does not matter if a terrible model has a thousand duplicates if none of those duplicates are ever selected, but if the best model has duplicates, then those duplicates will 'block' the selection of complementary models under the simple top-k recommendation (but not in recommendation boosting).

Chapter 8

Incomplete Ratings

In a typical recommender system, the items which users rate are not controlled by the system, and must simply be accepted. In contrast, in a model recommendation application, the system can decide not only how many models are rated on each task, but also *which* models are rated.

This chapter explores two possible benefits of being able to decide which items are rated. The first, recommendation from incomplete ratings, explores the question of how to make recommendation when the *ratings store* is incomplete, as well as quantifying the performance cost of sparse ratings information and whether it is worthwhile to trade a denser (more complete) ratings matrix for a larger but less complete one.

The second part of the chapter considers whether there are performance gains to be made from fine-grained control over which models are rated as the probe models. Instead of choosing random models to rate as the probes, is it possible to select models whose ratings will be more informative for predicting the ratings of the remaining models?

8.1 Recommendation from Incomplete Ratings

For simplicity the previous sections have assumed that the ratings matrices from which the recommendations do not have any missing (unknown) ratings. However, recommendation systems for consumers must deal with highly incomplete ratings data (e.g., each Netflix user rates only a small fraction of the 10,000+ movies in the database), and the collaborative filtering techniques used by these systems must likewise cope with this type of highly incomplete information.

8.1.1 Factorization with Incomplete Ratings

Factorization methods are the best posed to deal with incomplete ratings data, because their typical assumption of a low-rank ratings matrix makes theoretically well-posed factorization with incomplete ratings a possibility.

In the general case, this problem can be seen as one of *matrix completion*, in which the goal is to reconstruct a low-rank matrix (of possibly unknown rank) using only a small number of known entries of the matrix. This class of problems has been well studied, and there are strong theoretical guarantees in many cases. For example, provided that the known ratings are uniform

randomly distributed throughout the matrix, Candès and Recht [13] proved that a $n \times n$ matrix of rank $k << n$ can be *exactly* reconstructed from v visible entries if

$$v \geq Cn^{1.2}k \log n, \tag{8.1}$$

"most" of the time, for some constant C. This is not strictly possible for *all* such matrices, because (for example) if there are no visible entries at all in a row or column of the matrix, obviously it will be impossible to reconstruct that row or column.

However, for the purpose of recommendation systems, the ratings matrices are not actually low-rank; in fact, they are almost certainly full-rank simply due to the inherent noise produced by human provided ratings. Rather, these matrices can be seen, or more likely simply *assumed*, as approximately low rank, where most of the variation in the matrices is explained by a low-rank assumption.

Since the matrix is not exactly low rank, it obviously cannot be exactly reconstructed as the product of two low-rank matrices. Instead, the goal is to find rank k matrices U and V such that $UV \approx R$. Nevertheless, low-rank matrix completion algorithms can still be used in these situations, and some, such as OptSpace [44] have shown themselves to perform quite well even in the presence of noise [43].

In the collaborative filtering literature, factorization with incomplete ratings is rarely accomplished using low-rank matrix completion algorithms, but instead is usually implemented using either alternating least squares or gradient descent based techniques [107] including stochastic gradient methods for very large datasets (e.g., Netflix) [7]. Wu [107] presents a typical approach of this type. Since these approaches optimize the same objective function as SVD (*i.e.*, minimizing the squared error in the reconstructed matrix entries), results using these approximations are not presented here, but would be useful for very large scale applications.

8.1.2 Evaluating the Cost of Incompleteness

The accuracy penalty of using incomplete ratings is evaluated on the UCF50 dataset, with results shown in Fig. 8.1. OptSpace is the best performer, with GROUSE performing surprisingly poorly. Note that provided the density is larger than approximately 0.7, there is little penalty when using OptSpace for the factorization. Even for densities as low as 0.5, the accuracy penalty is quite modest. Interestingly, there appears to be a 'ledge' at around a density of 0.3, below which the accuracy of OptSpace quickly converges to that of mean imputation.

8.1.3 Store Size vs. Completeness

The key question with regards to incomplete ratings in model recommendation is whether it is better to invest the limited number of known ratings (model evaluations) into a smaller, but complete ratings matrix, or into a larger, but incomplete ratings matrix. For example, suppose that in an application with 1000 models the computation budget allows for $100,000$ model evaluations. Is it better to use those evaluations to construct a 1000×100 ratings store where every rating is known, or a 1000×1000 ratings store where only 10% of the ratings are visible? This question is unique to model recommendation, because in consumer item recommendation systems the system cannot force users to rate specific items in order to produce a "denser" matrix.

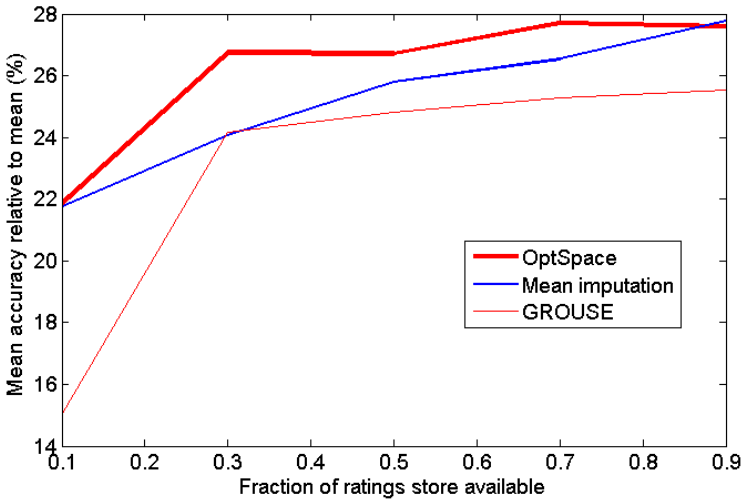


Figure 8.1: Effect on accuracy of incomplete ratings information in the store for factorization based recommendation on the UCF50 dataset.

This section tests this question using the UCF50 dataset that has been used in previous sections. While the previous sections used a complete 1000×1000 ratings matrix, or in Sec. 8.1.2 a 1000×1000 matrix with various levels of completeness, in this experiment the number of visible ratings is held constant at $100,000$, while the size of the ratings matrix is varied from 1000×100 at 100% visible to 1000×1000 at 10% visible. Which ratings are visible are chosen uniformly at random; unlike in consumer item recommendation systems where it is not reasonable to assume uniformly distributed known ratings, in model recommendation the system can indeed choose to evaluate ratings uniformly at random.

The quantitative results for this procedure using OptSpace and mean imputation can be seen in Fig. 8.2. GROUSE is omitted because it does not perform as well as OptSpace. Interestingly, it appears that there is no benefit from using larger, but sparser matrices, although there is a relatively large region where the results do not degrade either. From a range of 100% dense to approximately 50% dense, there seems to be a kind of conservation effect, where spreading the fixed number of known ratings out into a larger matrix neither improves nor degrades results. Once the density of visible ratings drops below approximately 50% visible, however, increasing the size of the matrix degrades results.

From a practical standpoint, then, the conclusion is that there is no benefit from incomplete ratings: if evaluating a complete ratings matrix is possible, then that is the preferred option. However, if evaluating a complete matrix is inconvenient, then provided the same number of entries are computed, a larger but incomplete matrix will perform nearly as well, provided that it is over 50% visible. However, the exact tradeoff is likely to be application specific.

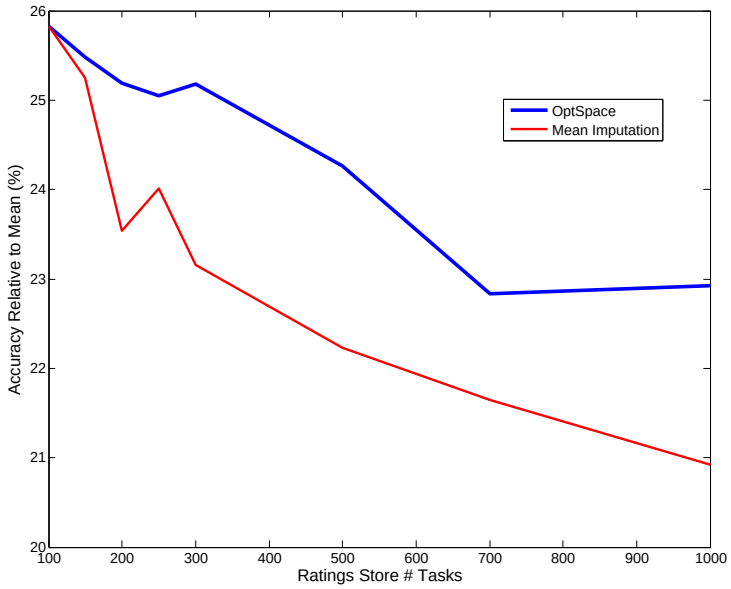


Figure 8.2: Effect of trading ratings store size vs. completeness for the UCF50 dataset.

8.2 Probe Selection

8.2.1 Optimal Design of Experiments

First we briefly summarize the setup for linear experimental design, as described by Melas [70] or Chaloner *et al*. [18].

Suppose that one wishes to engage in a number of experiments to determine the relationship between a dependent variable $y \in R$ and a vector of independent variables $x \in R^n$; that is, one wishes to find the function $y = f(x)$. In this thesis we only consider *linear* design of experiments (the reason for this limitation will become apparent once we link optimal design of experiments to collaborative filtering), in which $f(x)$ is a linear combination of n known basis functions

$$f(x) = \alpha_1 b_1(x) + \alpha_2 b_2(x) + \cdots + \alpha_n b_n(x), \tag{8.2}$$

with α_i being the coefficient of basis function b_i. This restriction is not as limiting as it may seem, since many useful functions (*e.g*., polynomials) can be represented as a linear combination of a small number of basis functions.

Now, if we have engaged in $m \geq n$ experiments, so that y_j is the result of each experiment's corresponding x_j, then the problem of estimating the αs is a simple linear problem, namely solving for

$$B\alpha = Y, \tag{8.3}$$

where B is an $m \times n$ matrix where $B_{ji} = b_i(x_j)$, and Y is the vector such that $Y_j = y_j$. The problem addressed by optimal design of experiments is *how do we choose the x_j so that we get the best estimate of the αs*? In optimal design of experiments, typically the continuous domain of x, R^n, is discretized into a finite *candidate set* of points at which experiments may be performed. Then, the question is which subset of candidate points to choose in order to get the best estimate of the regression coefficients α_i.

Now, suppose that we have k candidate points, then we can denote the full matrix of each basis evaluated at each candidate point as S, where $S_{ji} = b_i(x_j)$. Then, the optimal experimental design question is which subset of the rows to take (denote it S') to give the best estimate of α, when it is obtained by solving the linear least squares problem $S'\alpha = Y'$, where Y' is the corresponding subset of Y. This subset matrix is called the design matrix.

While many criteria have been proposed for optimal designs, a popular one is the D (or determinant) criterion, which seeks to maximize the determinant of the information matrix ${S'}^T S'$. Then, the optimal design is chosen according to

$$S^* = \underset{S'}{\operatorname{argmax}} \det({S'}^T S'). \tag{8.4}$$

Since this problem is non-convex, various algorithms have been proposed, such as greedily exchanging pairs of points from the candidate set into the design.

8.2.2 Relationship of Factorization Based Collaborative Filtering to Optimal Design of Experiments

The goal of factorization based collaborative filtering approaches is to represent user ratings of items and the dot products between user factors vectors and items factors vectors, so that a given item i's rating by user j is approximated by $r_{ij} = f_i \cdot u_j$, where f_i is the factors vector of item i, and u_j is the user factors vector of user j. Once these factors have been estimated for users and items, unknown ratings can be simply predicted. If the matrix of ratings is R, then this approach equivalently states that $R = FU^T$, where F is the matrix produced by concatenating all the item factors vectors together, one per row, and likewise U is the matrix of concatenated user factors vectors.

Given a new user with unknown user factors u^*, their user factors can be estimated as the linear least squares solution of

$$F'u^* = r^*, \tag{8.5}$$

where F' is the subset of rows of F corresponding to the items that the new user has rated, and r^* is the ratings the new user has given to that subset of items.

Now, typically in collaborative filtering the known ratings for a user are outside of the system's control, because users will rate what they rate. However, if users could be forced to rate certain items, it raises the question: which items should a user rate to produce the best estimate

of their user factors? A moment of reflection will reveal that this is exactly a linear optimal design problem, and the solution is that we should pick the subset of items the user should rate according to the d-optimality criterion, so that

$$F^* = \underset{F'}{\operatorname{argmax}} \det(F'^T F'), \tag{8.6}$$

where F' corresponds to a subset of z rows of F. In practice, we use simulated annealing to optimize for this selection, starting with a random design of z rows from the candidate set of all rows of F, and where each annealing step exchanges an unused row from F for one in F'.

8.2.3 Evaluating Probe Selection

We evaluate how well the d-optimality selects probes for the UCF50 application, using simulated annealing to optimize for the d-optimality criterion. Note that if fewer probes are selected than the number of factors k in the factorization, then it will not be possible to optimize for the probe selection, since regardless of the selected probes, $\det(F'^T F') = 0$. Since we use 64 factors for this application, the minimum number of probes that can be selected through design of experiments is 64. These results can be seen in Fig. 8.3.

Note that although the performance improvement at any fixed number of probes seems to be marginal, in terms of the number of probes required to reach a given level of accuracy, the probe selection essentially halves the number of probes that must be evaluated. For example, recommendation with probe selection does almost exactly as well with 100 probes as recommendation with random probes does with 200 probes. Likewise, with probe selection almost same accuracy can be achieved by trying merely 500 as by performing the recommendation with all 1000 models as probes. Compared to direct selection, recommendation with probe selection does better with 100 probes (10% of the total) than direct selection does trying every model in the library, which is not true of recommendation with random probes.

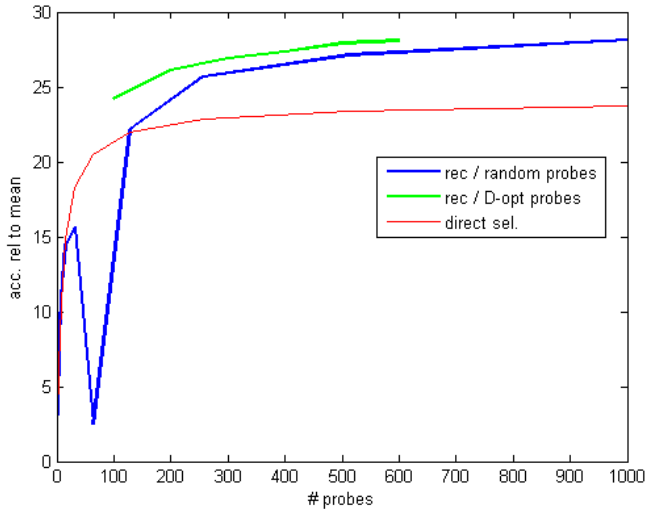


Figure 8.3: Probe selection on the UCF50 application using the D-optimality criterion.

Chapter 9

Sequential Selection

The previous chapters considered ratings to be essentially immutable, in the sense that once a rating is estimated for a model on a task, it cannot be changed or improved, so that an inaccurately estimated rating is consigned to remain that way forever. This is the case for classifiers rated by their accuracy on fixed tasks, since a (deterministic) classifier will always obtain the same accuracy on the same dataset. Even though that classifier's estimated accuracy on the training set may be a poor measure of its hidden accuracy on the test set, running it additional times on the same training set will not provide any additional information.

However, in some applications it may be possible to repeatedly rate the same model on the same task and thereby actually improve the estimate of the rating. This is especially likely to be the case when the task is not a fixed dataset, but rather a process.

For evaluation and illustrative purposes we consider the following simulated (yet plausible) scenario: a fleet of simple, state-machine driven robotic vacuum cleaners are purchased by individual users and set to work in their respective rooms whose layouts are unknown. If we consider the robot of a specific user then each day the robot may pick a policy (a state machine) to run from a library of such state machines and, after that day's excursion, it is told how much of the room it covered. Additionally, it may also access a central database to query how well the state machines in the library performed on *other* room layouts. The objective is to find a high coverage state machine as quickly as possible. Complicating the problem is the fact that each run starts in a randomized position in the room and that the state machines may be randomized themselves; thus, the coverage returned at the end of the run is only a noisy estimate of the true quality of that particular state machine (see Fig. 9.1 for an example of a simulated run of a state machine on a room).

There are two broad ways of approaching the problem: first, one might consider the multi-task aspect and how to use the reported coverages of the state machines across different unknown room layouts to improve the state machine choice for each individual room. The model recommendation approach detailed in the preceding chapters falls under this approach.

Alternatively, the problem can be viewed as a k-armed bandit [83]: if there are a finite number (k) state machines, then each state machine can be seen as an arm of the bandit. Each day, the robot can pick an arm to pull, and receive a randomized payout according to that arm's distribution. The goal is to converge on the arm with the highest long-term payout (expected payout) as quickly as possible. However, in the traditional multi-armed bandit formulation, the

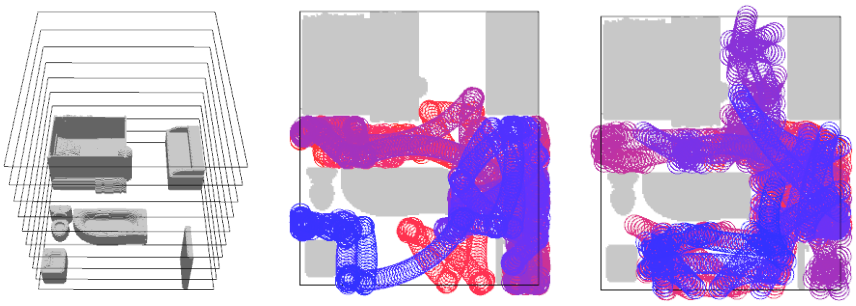


Figure 9.1: Runs of two different state machines on the same room layout. Time is represented by both the vertical axis and color (red→blue). The average best state machine across all rooms (left) takes large sweeping curves that are good for open rooms, but which makes it navigate tight spaces less effectively. In contrast a state machine that is good for navigating tight spaces (right) does well on this room but poorly overall. The problem is how to effectively pick the state machine suited for each room.

algorithm has to try each arm at least once to get some estimate of its quality [8]; if the state machine library is large, then even trying each option once could take prohibitively long.

While model recommendation is able to use the collective knowledge of the robots in order to collaboratively select policies, it makes those selections offline without tackling the sequential nature of the choices and the exploration-exploitation tradeoff. Conversely, interpreting the problem as a k-armed bandit addresses the sequential nature and fundamental exploration-exploitation tradeoff, but does not take advantage of the shared knowledge between robots operating on parallel tasks.

In joint work with Michael Furlong, we merge the two interpretations and demonstrate how collaborative filtering can be applied to the k-armed bandit problem to produce a hybrid technique that outperforms both of its constituents.

9.1 Related Work

There is a significant body of work on multi-task policy optimization focusing on learning good policies for Markov Decision Processes (MDPs) or Partially Observable Markov Decision Processes (POMDPs). However, these types of approaches tend to be restricted to relatively small grid worlds (30×30) because the approaches become intractable for large state spaces [56, 84, 99, 104].

Attempts have been made to deal with large state spaces in various ways; for example, Sallans *et al.* [89] try to learn an approximator function for the value of a state-action pair. However, it may still take thousands of iterations to learn a good policy. An early work by Parr and Russell [78] considered reinforcement learning over partially specified state machines that could 'call' each other as sub-modules; this allowed larger state spaces (60×60), but the larger grid-world maps relied on explicitly repeated obstacles.

On the other end of the spectrum, rather than attempting to generate policies from a pure learning perspective, there has been much work on complete coverage problems for mobile robots [24, 106]. However, these works tend to require accurate localization of the robot whereas

we consider a largely blind robot with only rudimentary sensing. The capabilities of our robot more closely resemble the early state-machine driven work of MacKenzie and Balch [63].

As mentioned previously, the repeated runs and selections of state machines can be seen as a type of bandit problem, which were introduced by Robbins [83] as a means of sequentially determining which experiments to run. In our setting the experiments in question are trials of robot state machines on room layouts. Lai and Robbins [51] introduced idea of valuing experiments by summing the mean and standard deviation of previously observed rewards as a means of addressing the exploration/exploitation trade off. Similarly, Schmidhuber [91] used the confidence in the prediction of rewards as the measure of value for experiments. Auer *et al*. [4] refined the approach of Lai and Robbins further with the Upper Confidence Bound (UCB) algorithm. UCB addresses the exploration/exploitation problem by valuing experiments with an index that increases with time since the last evaluation of a given experiment in addition to the expected reward of a given experiment. These approaches need only know how to evaluate the result of a given experiment and need not understand its operation, which make them particularly suitable for the task of evaluating robot state machines, which for the purpose of our technique can be considered black boxes.

9.2 Neighborhood Collaborative Filtering with Variance Estimates

While previous chapters have concentrated on factorization based collaborative filtering, for this scenario the simple k-nearest-neighbor recommendation technique provides better results. The set of state machines whose coverages have been measured on the current room is denoted by C; these are the entries in the current room's ratings vector that are known. Based on these visible dimensions of the ratings vector, the room's ratings vector can be compared to the ratings vectors of the rooms in the database (the columns of R). Then, let N be the set of k nearest neighbors of the columns of R according to this ratings vector comparison (simple l^2 distance). The predicted mean coverage and variance for a model can then be computed according to Alg. 5. Predicted variances can also be computed for factorization models, but the method is not as straightforward. While Koren [46, 48] suggests that the correlation coefficient between ratings might be used to determine the neighbors, in practice we find that the simple Euclidean distance outperforms the correlation coefficient.

This can be seen as a special case of the neighborhood techniques used in the literature, whose complexity derives mainly from having to deal with missing ratings in the store R [5, 46] and scale to very large datasets [48].

9.3 Evaluation Scenarios

Strategies are evaluated on two scenarios common to a wide range of applications: fixed horizon and indefinite horizon. The fixed horizon simulates a situation where a robot is allotted a fixed 'learning' period, after which it must commit to what it believes to be the 'best' policy. The

Algorithm 5 k-Nearest-Neighbor Collaborative Filtering with Variance Predictions

function PREDICTEDCOVERAGE(C, m_i)
  $N \leftarrow \text{Neighbors}(C, R, k)$
  return $\mu_p = \frac{\sum_{q \in N} r_{iq}}{|N|}$
end function
function PREDICTEDVARIANCE(C, m_i)
  $N \leftarrow \text{Neighbors}(C, R, k)$
  $\mu_p \leftarrow \text{PredictedCoverage}(C, m_i)$
  return $\sigma_p^2 = \frac{\sum_{q \in N} (r_{iq} - \mu_p)^2}{|N|-1}$
end function

indefinite horizon experiments mimic a scenario where a robot is expected to continually improve in performance from day 1, without any learning or 'burn in' period.

In a fixed horizon scenario, a strategy is told how many state machine evaluations (days) it is allotted (the horizon). Then, each day the strategy is allowed to choose one state machine which runs for 3000 ticks on that room, and the room coverage obtained by the chosen state machine is reported back. At the end the strategy must commit to a final choice of state machine, and that final choice is run for 10 trials (3000 steps per trial) on the room. The strategy's performance on a room is measured as the ratio of that average coverage on the final ten trials to the coverage of the best performing state machine on that room. This coverage fraction is averaged across all 500+ rooms in the test set to produce the strategy's aggregate score.

In the indefinite horizon scenario, a strategy is run on a test room for a number of days which is not revealed to it. Unlike the fixed horizon experiments where only the final choice was evaluated, in an indefinite horizon experiment the strategy's daily choices are evaluated. The performance of a strategy is measured as the average accumulated fractional regret. For each day's performance the fractional regret is computed as $\frac{coverage_o - coverage_t}{coverage_o}$, where $coverage_o$ is the coverage obtained by the optimal choice of state machine for that room and $coverage_t$ is the measured coverage of a state machine on day t. The average accumulated fractional regret for a room on day t is computed as $\frac{1}{t} \sum_{i=1}^{t} \frac{coverage_o - coverage_i}{coverage_o}$. Note that a strategy does not know its regret while running.

Successfully balancing the exploration/exploitation trade-off involves minimizing a combination of the time it takes to converge to an optimal choice and the regret of that choice, as it can be better to converge more quickly to a higher regret than to take a long time to obtain only slightly better asymptotic performance.

9.4 Strategies

Let M be the set of all models (state machines) in the library. For the fixed horizon experiments, the strategy is initially told the duration of the exploration period ($t_{\max}$).

Strategies may use two pieces of information about each state machine: its MeasuredCoverage, which is the mean coverage (and variance) obtained by that state machine on the current room over all the times it was chosen (if a state machine has never been chosen, its measured cover-

age is undefined). Strategies may also use the $\mathrm{PredictedCoverage}(C, m)$ of a state machine m, which is the coverage predicted by the recommendation system given the measured coverages of the chosen state machines in the set C. Note that if $C = \emptyset$, that is, if a prediction is requested without having actually tried any state machines on the current room, then the recommender returns the mean coverage of the requested state machine across the database.

Each day t, a strategy may choose one state machine $m_t \in M$ from the library, after which the measured coverage of that state machine is available to the strategy. For the fixed horizon evaluation, at the end the strategy must commit to a final state machine. We explore six different strategies:

Algorithm 6 Random search fixed-horizon algorithm

function INITIALIZE($M, t_{\max}$)
  $P \leftarrow M$
  $C \leftarrow \emptyset$
end function
function CHOOSESM(t)
  $m_t \leftarrow \mathrm{RandomChoice}(P)$
  $P \leftarrow P - m_t$
  $C \leftarrow C \cup (m_t, \mathrm{MeasuredCoverage}(m_t))$
end function
function CHOOSEFINALSM(t)
  $m_f \leftarrow \arg\max_{m \in C} \mathrm{MeasuredCoverage}(m)$
end function

9.4.1 Random

At each iteration, choose a random state machine that has not been previously chosen, and evaluate its coverage. As the final choice, choose the state machine with the highest measured coverage (Alg. 6).

9.4.2 Batch Recommendation

For the first αT days, choose random state machines to evaluate. Then, using the measured coverages of those state machines, use recommendation to predict the coverages of the remaining state machines. For the remaining $(1 - \alpha)T$ days, evaluate the unevaluated state machines in decreasing order of their predicted coverages. As the final choice, choose the state machine with the highest measured coverage. See Alg. 7.

9.4.3 Greedy Recommendation

Each day, predict the coverages of all the state machines, using the coverages of the state machines evaluated so far as input. Choose the state machine with the highest predicted coverage

Algorithm 7 Batch recommendation fixed-horizon algorithm

function INITIALIZE($M, t_{\max}$)
  $P \leftarrow M$
  $C \leftarrow \emptyset$
end function
function CHOOSESM(t)
  if $t < \alpha T$ **then**
    $m_t \leftarrow \text{RandomChoice}(P)$
    $P \leftarrow P - m_t$
    $C \leftarrow C \cup m_t$
  else
    $m_t \leftarrow \arg\max_{m \in P} \text{PredictedCoverage}(C, m)$
    $P \leftarrow P - m_t$
    $C \leftarrow C \cup (m_t, \text{MeasuredCoverage}(m_t))$
  end if
end function
function CHOOSEFINALSM(t)
  $m_f \leftarrow \arg\max_{m \in C} \text{MeasuredCoverage}(m)$
end function

Algorithm 8 Greedy recommendation fixed-horizon algorithm

function INITIALIZE($M, t_{\max}$)
  $P \leftarrow M$
  $C \leftarrow \emptyset$
end function
function CHOOSESM(t)
  $m_t \leftarrow \arg\max_{m \in P} \text{PredictedCoverage}(C, m)$
  $P \leftarrow P - m_t$
  $C \leftarrow C \cup (m_t, \text{MeasuredCoverage}(m_t))$
end function
function CHOOSEFINALSM(t)
  $m_f \leftarrow \arg\max_{m \in C} \text{MeasuredCoverage}(m)$
end function

that has not already been evaluated. As the final choice, choose the state machine with the highest measured coverage.

9.4.4 Upper Confidence Bound Bandit (without recommendation)

Algorithm 9 UCB Bandit

function INITIALIZE($M, t_{\max}$)
  $\mu \leftarrow \emptyset$
  $\sigma \leftarrow \emptyset$
  $n \leftarrow \emptyset$
end function
function CHOOSESM(t)
  if $|\mu| < |M|$ **then**
    $m_t \leftarrow m : m \in M \bigwedge m \notin \mu$
    $\mu_m \leftarrow \text{MeasuredCoverage}(m_t)$
    $n_m \leftarrow 1$
  else
    $m_t \leftarrow \arg\max_{m \in M} \mu_m + \sigma_m + \sqrt{2\frac{\ln t}{n_m}}$
    $n_m \leftarrow n_m + 1$
  end if
  $c_t \leftarrow \text{MeasuredCoverage}(m_t)$
  $\mu_m, \sigma_m \leftarrow \text{UpdateStats}(\mu_m, \sigma_m, c_t)$
end function

The bandit uses the Upper Confidence Bound decision [4] rule to select the next state machine and has no input from the predictions of the recommender function. By necessity this strategy must first try every state machine at least once before it can begin to determine which state machine provides the best coverage. The strategy then computes a score for the state machines based on a running sample mean and sample standard deviation which are in turn derived from observations of the state machines running in a given room. See Alg. 9.

9.4.5 Bandit Recommendation

This version of the bandit uses the recommender function and the history of observed coverages to estimate the mean and variance of the coverage for the different state machines. By relying on the recommender function for these quantities the bandit need not engage in lengthy periods of time exploring the space of models. See Alg. 10.

9.4.6 Neo-Bandit Recommendation

The Neo-Bandit is like bandit recommendation except that it does not fully trust either the predicted coverages or the measured coverages, but instead takes a minimum-variance weighted

Algorithm 10 Bandit recommendation

function INITIALIZE($M, t_{\max}$)
$P \leftarrow \{\text{PredictedCoverage}(\emptyset, m) : m \in M\}$
$C \leftarrow \emptyset$
$n \leftarrow \{1 : m \in M\}$
end function
function CHOOSESM(t)
$m_t \leftarrow \arg\max_{m \in P} \text{PredictedCoverage}(C, m) + \text{PredictedDeviation}(C, m) + \sqrt{2\frac{\ln t}{n_m}}$
$n_m \leftarrow n_m + 1$
$C \leftarrow C \cup (m_t, \text{MeasuredCoverage}(m_t))$
end function

Algorithm 11 Neo-Bandit

function INITIALIZE($M, t_{\max}$)
$P \leftarrow \{\text{PredictedCoverage}(\emptyset, m) : m \in M\}$
$C \leftarrow \emptyset$
$n \leftarrow \{1 : m \in M\}$
end function
function CHOOSESM(t)
$m_t \leftarrow \arg\max_{m \in P} \text{BestEstimateUpperBound}(m) + \sqrt{2\frac{\ln t}{n_m}}$
$n_m \leftarrow n_m + 1$
$c_t \leftarrow \text{MeasuredCoverage}(m_t)$
$\mu_m, \sigma_m \leftarrow \text{UpdateStats}(\mu_m, \sigma_m, c_t)$
end function
function BESTESTIMATEUPPERBOUND(m)
$\mu_p \leftarrow \text{PredictedCoverage}(C, m)$
$\sigma_p^2 \leftarrow \text{PredictedVariance}(C, m)$
if $n_m < 2$ **then**
$\mu_b \leftarrow \mu_p$
$\sigma_b^2 \leftarrow \sigma_p^2$
else
$\alpha = \frac{\sigma_p^2}{\sigma_p^2 + \sigma_m^2}$
$\mu_b = \alpha \cdot \mu_m + (1 - \alpha) \cdot \mu_p$
$\sigma_b^2 \leftarrow \alpha^2 \sigma_m^2 + (1 - \alpha)^2 \sigma_p^2$
end if
return $\mu_b + \sigma_b$
end function

mean of the two to produce a best-estimate of each state machine's mean coverage (with uncertainty). Note that the algorithm uses the *variance of the sample mean* for the measured coverage variance (*i.e.*, as the number of measured samples goes to infinity, the uncertainty on the measurement should go to zero). See Alg. 11.

9.5 Incomplete Database

So far we have operated under the assumption that the database of coverages of the state machines on other rooms is complete, *i.e.*, dense. However, in practice it will not be possible under realistic conditions to get such a database where every user has rated all the models in the database on their room, but will have instead only rated a fraction of the models. To simulate this condition, we corrupt the coverages database so that only 25% of the coverages are visible, ranging from a minimum of 29 rated models to a maximum of 73 (out of 200) on a single room.

Although many collaborative filtering methods natively deal with incomplete ratings, for simplicity we explicitly attempt to reconstruct the database's missing ratings from the visible ones. This allows us to use the previously presented algorithms unmodified. We use the OptSpace low-rank matrix completion algorithm [44] to complete the missing ratings.

9.6 Results

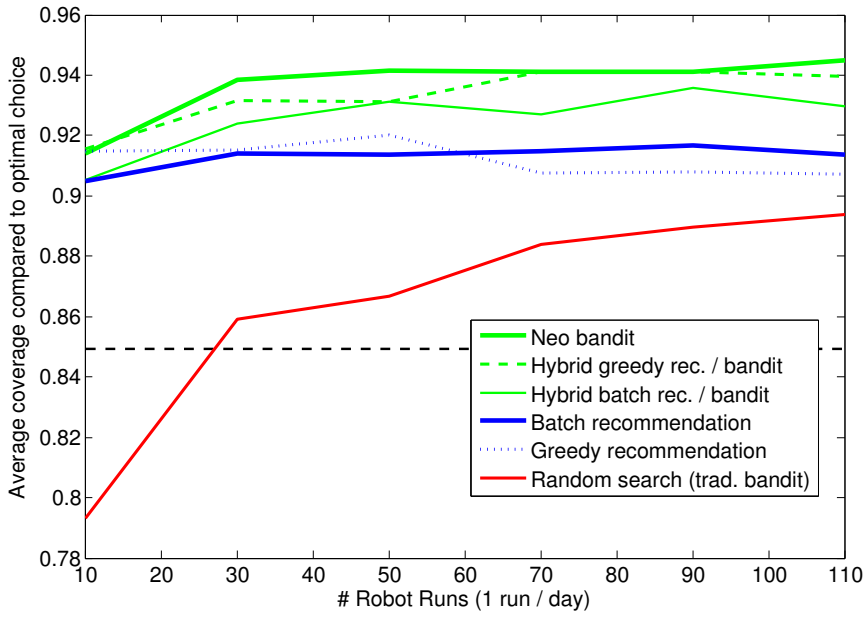


Figure 9.2: Results for the fixed horizon experiments. The dashed line is the coverage that would be obtained by picking the overall best (average best) state machine (0.85). Note that a traditional bandit must try each arm (model) at least once, and since the time limit for these results is less than the number of models (200), the traditional bandit degenerates to random search.

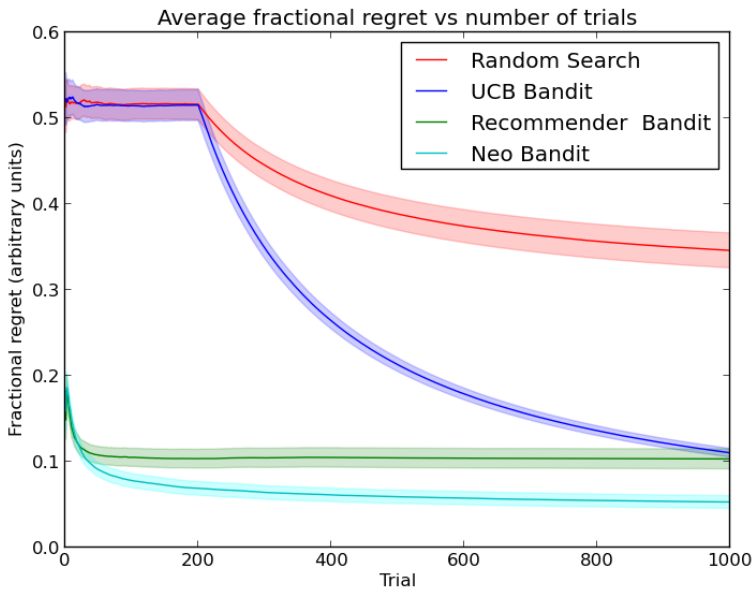


Figure 9.3: Results for the indefinite horizon experiments. The solid lines represent the regret averaged over the test rooms the strategies were evaluated on. The shaded region represents a 95% confidence interval around that mean. The UCB bandit spends the first 200 days examining the different state machines and is not significantly different from random search during that time. Once the bandit completes its initial exploration over the state machines its performance begins to converge towards the performance of the other bandit algorithms. The Recommender and Neo-Bandit algorithms both converge very quickly to a low average regret. The Neo-Bandit converges to a much lower average regret and with a tighter confidence interval. It is clear to observe that the Neo-Bandit is the superior algorithm to employ in this setting.

Results for the fixed horizon experiments can be seen in Fig. 9.2. Although the pure recommendation strategies (blue lines) show a clear dominance over the random search / traditional bandit, they show little improvement with larger evaluation periods, and the greedy recommendation actually gets worse in performance for horizons beyond 50 evaluations / days. This is likely because the pure recommendation strategies quickly converge to relatively stable (but possibly inaccurate) predicted coverages. Combining the recommendation with the bandit strategies obtains the best results (green lines). Note that unlike the pure recommendation strategies, these continue to improve with larger evaluation periods, because the bandit continues to evaluate even the arms (models) that have already been tried. Interestingly, these results show a clear preference for greedy type algorithms: the hybrid greedy recommendation bandit (which acts as greedy recommendation for the first half of the evaluation period, then acts as the recommendation bandit) outperforms the hybrid batch recommendation, but is in turn outperformed by the neo bandit. Indeed, the top performer (neo bandit) can be seen as the most aggressively greedy algorithm: unlike the 'greedy' recommendation, which even in the greedy stage enforces the constraint that it must try *different* state machines, the neo bandit can repeatedly pick exactly the same model day after day if it has the highest upper coverage bound.

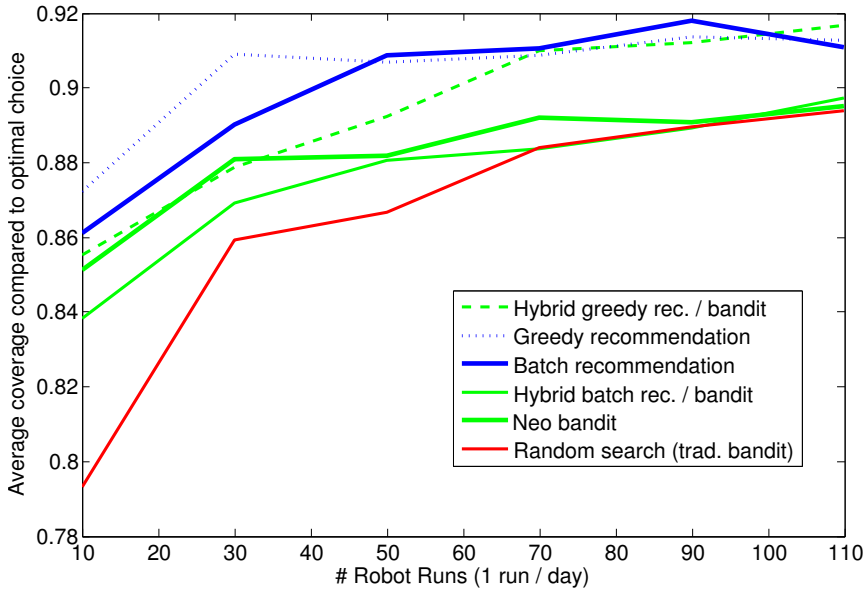


Figure 9.4: Results for the fixed horizon experiments, using an incomplete ratings (coverage) database.

The result of choosing the average best state machine according to to the training / database rooms is indicated as the dashed line (0.85). Surprisingly, the average best state machine fares relatively poorly even compared to random search: trying a random state machine each day for a month and then selecting the best does better than committing to the single average best state machine.

The results for the indefinite horizon experiments can be seen in Fig. 9.3. During their exploration phase the random search and the UCB bandit are not statistically different from each other. It is only after the initial exploration phase when the UCB bandit continues to refine its estimates of coverage for the state machines that the differences in performance begin to emerge. Similarly, the Recommender Bandit and the Neo-Bandit are initially not statistically different from each other, and both outperform the other two strategies well before the random search and UCB bandit finish their initial exploration. The UCB bandit had not converged before the end of the indefinite horizon experiment while the Recommender and Neo-Bandits have. Clearly using prior knowledge significantly reduces time to convergence.

By acting as a variance weighted combination of the UCB and Recommender bandits, the Neo-Bandit outperforms both, as can be seen in Table 9.1. The Neo-Bandit had both the lowest average regret and the smallest standard error, approximately half that of the next best algorithm, the Recommender bandit. Further, the Neo-Bandit converges faster than the UCB bandit, which does not converge before the end of the indefinite horizon experiment. All of the bandit algorithms produce more reliable results than the random search, as implied by the standard error.

Results using the incomplete ratings database can be seen in Fig. 9.4. The recommendation based strategies still outperform the typical bandit, however, the difference is less dramatic. The interesting result here is how the ordering of the strategies has been upset: whereas the fixed horizon results with the complete database showed a clear preference for greedier algorithms, here the reverse is true, since the best results (batch recommendation and greedy recommendation)

Table 9.1: The final average regret and standard error of regret for the indefinite horizon experiment (arbitrary units).

Algorithm	Final fractional regret	Final standard error
Random Search	0.346	0.0104
UCB Bandit	0.110	0.0026
Recommender Bandit	0.103	0.0059
Neo-Bandit	0.053	0.0038

are limited in their greed. This is likely because with an incomplete and reconstructed ratings database, there is significantly more error in the predictions, and so trusting the predictions more heavily (as the bandit recommendation strategies do) can lead down an unfruitful path. In contrast, even the 'greedy' recommendation forces itself to explore different state machines during the evaluation period

9.7 Conclusions

This chapter has demonstrated a method for quickly selecting a robot policy (state machine) to maximize room coverage for a robot from a library of such policies. However, this method is general to robotics applications where directly learning a policy is difficult yet a library of reasonable policies can be generated, and where it is possible to quantify the quality of those policies on individual instances of the problem. Since the method considers the actual policies as black boxes and only directly considers the *ratings* of the policies, the underlying policies can take virtually any form: state machines, neural networks, or even hand-coded programs are all possibilities. The method could be applied to selecting gaits for different terrains, strategies for robotic soccer teams, or even grasps for different objects.

Furthermore, although results are presented on simulations, the specific problem evaluated against is not a completely abstracted toy problem, as there are only a few practical hurdles involved with implementing the method on real cleaning robots. Foremost is the question of how coverage might be effectively measured in a real robot: while outfitting a robot with motion-capture markers or the like would certainly solve the problem, once accurate localization is a possibility, simply directly mapping the room becomes an option. One possibility is that a human being could directly rate the robot's performance on a simplified 'cleanliness' scale. These human-provided ratings would be extremely noisy, but collaborative filtering techniques were developed to deal with noisy human ratings, so there is still hope that in aggregate there could be enough information to make useful recommendations.

Chapter 10

Conclusions and Future Directions

This thesis has explored the connection between collaborative filtering based recommender systems and the problem of selecting "models" for tasks within vision and robotics applications. Whereas a typical consumer recommendation system will use a user's ratings of items to recommend other items that user might enjoy, in model recommendation the ratings of models on a task are used to recommend other models that are likely to work well for that task. These notions of "model", "task", and "rating" are quite general, and we show that they can be easily adapted to accommodate a wide range of applications.

10.1 Summary of Contributions

This section summarizes and re-iterates the contributions that have been presented in this thesis.

10.1.1 Trajectons and Feature Seeding

We have presented one of the first methods based on using tracked keypoint trajectories in a bag-of-words (BoW) framework, as well as a powerful method for representing spatial relationships between generic sparse BoW features. Furthermore, we have improved on the quantization scheme for the trajectory features by introducing a method for selecting a "generally good" quantization scheme for human motions derived from motion capture data. We demonstrate that the quantization scheme "seeded" from semi-synthetic rendered motion capture data outperforms hand designed algorithms and feature selection techniques on an action recognition problem.

10.1.2 Model Recommendation

We have shown that, surprisingly, it is indeed possible for a recommender system to pick a better model for a task than even trying every model on that task, and that the effect is common across a wide range of datasets and applications, including action recognition and 3D model matching case. For the 3D model matching application, we also demonstrate how to adapt model recommendation to efficiently search over models when computing the rating of a model on a task is computationally expensive.

10.1.3 Ensemble Recommendation

We have extended the recommendation concept beyond the single-item recommendations produced in consumer recommendation systems. In the ensemble recommendation problem, the objective is to jointly recommend a set of classifiers to be combined into a stronger one. We introduced the recommendation boosting algorithm, and demonstrated that it is resilient against redundant or duplicated models in the library. We also empirically measured the redundancy of the model libraries for different applications, and found that they varied from almost no redundancy, to over 100-fold redundancy.

10.1.4 Sequential Selection

We have extended the model recommendation concept to a sequential selection problem in a multi-armed bandit framework, where each model in the library can be seen as a bandit arm. Using the recommendation bandit algorithm, good results can be obtained much faster than by either a traditional bandit or by the naive application of model recommendation to the problem. We demonstrate this algorithm on a robotic control problem, where the objective is to choose the best state machine to drive a robot to maximize floor coverage in an unknown room layout.

10.1.5 Incomplete Ratings

On a practical front, we have investigated a number of questions relating to how the ratings store should be constructed and the probes selected, and we suggests a number of good practices for applying collaborative filtering to classifier or other model libraries. First, it is better to have a small and more complete ratings store than a larger but less complete one. A real world application may have less control over this, since end-users might not be pleased to have their devices "wasting" computation time evaluating models other than the predicted best one for their particular task. However, as the recommendation bandit results show, a method which simply focuses on what it considers to be the single best prediction without exploring the rest of the space will not do as well as one that is more exploration focused.

10.2 Avenues for Future Exploration

While this thesis has thoroughly covered the basics of model recommendation, it has also raised a number of interesting questions and directions that future research and thesis might consider. Looking forward, we identify three key areas for exploration.

10.2.1 Adapting Recommended Models

We have been careful in this thesis to consider models as opaque black boxes, where they can only be inspected by rating them on different tasks. This provides a very general basis from which to adapt the model recommendation idea to different applications. In the case of classification, however, there are a number of interesting domain adaptation techniques which might

be applied in conjunction with model recommendation to not merely recommend a single model or ensemble of models, but to actually break apart the models and re-tailor them to the specific task.

For example, in work by Fergus *et al*. [35], training samples are shared between different classification tasks by weighting them according to their semantic similarity as measured by the class labels' distance in WordNet. Instead of relying on this type of external metadata (the class labels and their distance in WordNet), model recommendation could be used as the similarity measure. This could be done in many ways; for example, task-task neighborhood recommendation could be used to find the neighbors of a task and then share training samples among tasks according to their neighbors. Another, more interesting, possibility is that training samples might be shared not from task to task, but from model to task. That is, the predicted accuracy of a model on a new task could be seen as a similarity measure between that model and task, and then the training samples used to train the models transferred and weighted according to the predicted accuracies of their models.

In a more general sense, there is a great deal of room for exploration in using the recommended models as starting points for further adaptation or transfer to the new task.

10.2.2 Ratings Store Growth

This thesis has largely considered the scenario where the ratings store has already been established, but a problem facing a real-world application is how to bootstrap the creation of that ratings store. One possibility which has been touched on in this thesis is to use synthetic data to produce synthetic tasks. Another is to allow the ratings store to simply grow organically and improve in quality over time. In that case, an interesting question is how probe selection techniques will interact with the growing ratings matrix. In the case of random probe selection, all is well, because most of the matrix completion methods assume that ratings are missing according to a uniform random distribution. However, if something like probe selection or a recommendation bandit is used, it is unclear how these two aspects (the matrix completion and the probe selection) will interact.

10.2.3 Real Applications

One main future direction is clear: while this thesis has used existing datasets to *evaluate* model recommendation, to truly develop the ideas they must be implemented in a *real* application. Just like collaborative filtering as a field did not really grow until Netflix made public their real user ratings data, what model recommendation needs to truly thrive is likewise a real application with real model performance data, as measured on real end-user tasks. That is to say that the application cannot be merely slicing and dicing existing or future datasets into tasks, but must actually be composed of tasks of true interest to end-users.

The main limiting factor in a real application is likely to be how the ratings are computed; in this thesis, the ratings have largely been accuracies, which require some amount of labeled training data to evaluate. A key question for a real application is how to solicit these annotated training samples without unduly burdening the user, or how to avoid needing labeled training data altogether.

Interestingly, there are a number of recent approaches that suggest that it may in fact be possible to evaluate how well many classes of algorithms (such as pose estimators [40]) perform without having the underlying ground truth, such as in Aodha *et al*. [62], Alt *et al*. [2], and Jammalamadaka *et al*. [40].

Alternatively, the ratings might not be automatically computed ratings at all, but qualitative user-provided ratings of how "well" the system is working. Since the notion of a "rating" is very general, there is broad room for interpretation of how these ratings should be measured in a real application. Perhaps, as was the case with the Netflix prize, there is reason to hope that this real application will emerge as the intended or unintended product of the type of data that companies are already collecting.

10.3 Concluding Thoughts

Collaborative filtering addresses the problem of how to choose, whether that manifests as a choice of which movie watch or which state machine to use to control a robot. The key insight of collaborative filtering is that this choice is best made not according to meta-data about the options (such as information about movie genre, cast, or director), but by learning the patterns and correlations between the *ratings* of those options across different users. That is to say, a user's movie preferences are better represented through their ratings of movies than by their stated genre affinities or the like. This thesis has taken that insight and adapted it to the analogous problem of choosing between options (such as different trained classifiers) for computer vision and robotics tasks.

However, to reach this insight, collaborative filtering had to already have abandoned the notion of "best", to discard the idea that there is a singular ideal ranking of items to be prescribed for all users. For movies, this was not much of a leap, as it has long been recognized (as in the well-known Latin saying *de gustibus non est disputandum*), that all tastes are subjective opinions and beyond argument or ranking. It was already clear that there was no "best" movie in general, only best movies for individual users. Yet in computer vision it often seems that we still cling to this notion of best, that there is a best face detector or a best action recognition algorithm and that we might ferret out this best by the broad application of benchmark datasets. After all, how many papers have boldly claimed to have improved on state of the art (with the implication that this improvement is general) based on a few percentage point gains on just a few key datasets?

The point is not to deny that currently there is still ample room for general improvement in most areas of computer vision, but to encourage consideration of what will happen when this era of general gains comes to an end and gains on one dataset or task come at the expense of performance on others. Once we have eliminated all the methods and algorithms that are strictly worse than others on all datasets and tasks, it is unlikely that we will be left with a single victor, but instead we are likely to find ourselves with a collection of methods which beat one another in various conditions and situations but out of which none can be unequivocally said to be the best.

Looking forward to that scenario, what this thesis has demonstrated is that this apparently simple selection problem is still harder and subtler than it appears, and that even when faced with such a seemingly simple choice there is still considerable opportunity for overfitting. Because of this possibility of overfitting, by treating the selection as a recommendation problem,

the ‘preferences’ of different tasks can collaborate to produce recommendations for individual tasks that outperform the isolated direct selection on each task of the *apparently* best performing option.

But, on a more speculative note, this thesis can also be considered as a step towards automating the very process of engineering specialized computer vision or robotics systems for different tasks. Today, a person wanting to build a vision system for a particular task they want solved would likely appeal to the literature to find similar applications and the algorithms employed therein. Based on which algorithms seem to be the most common, or best performing, or even simplest to implement, or some combination thereof, the person might *try* a few algorithms on their problem and select the one that appears to do the best as measured by the metric that is most relevant for their problem. To do this well requires an intimate familiarity with computer vision to be able to sort through the literature to determine what is trustworthy and relevant to the particular task at hand, and then to interpret the performance of the implemented methods.

Indeed, when phrased in such a manner, being a computer vision expert might be seen as just being a human recommender system for computer vision algorithms! While committing to such speculation is perhaps beyond the scope of this thesis, there is a certain satisfying irony in the thought that one day computer vision researchers might abstract and automate away their own expertise.

Chapter 11

Publications

The publications which comprise this thesis are listed below.

- P. Matikainen, M. Hebert, and R. Sukthankar
  Representing Pairwise Spatial and Temporal Relations for Action Recognition
  Proceedings of European Conference on Computer Vision, 2010

- P. Matikainen, R. Sukthankar, and M. Hebert
  Feature Seeding for Action Recognition
  Proceedings of the International Conference on Computer Vision, 2011

- P. Matikainen, R. Sukthankar, and M. Hebert
  Model Recommendation for Action Recognition
  Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition, 2012

- P. Matikainen, R. Sukthankar, and M. Hebert
  Classifier Ensemble Recommendation
  Proceedings of the ECCV Workshop on Web-scale Vision and Social Media, 2012

- P. Matikainen, P. Furlong, R. Sukthankar, and M. Hebert
  Multi-armed Recommendation Bandits for Selecting State Machine Policies for Robotic Systems
  In submission to the IEEE International Conference on Robotics and Automation, 2013

Bibliography

[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. 2.3

[2] N. Alt, S. Hinterstoisser, and N. Navab. Rapid selection of reliable templates for visual tracking. In *CVPR*, 2010. 2.4, 10.2.3

[3] A. Argyriou, T. Evgeniou, and M. Pontil. Multi-task feature learning. In *NIPS*, 2007. 2.3

[4] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2):235–256, 2002. 9.1, 9.4.4

[5] R. M. Bell and Y. Koren. Scalable collaborative filtering with jointly derived neighborhood interpolation weights. In *ICDM*, 2007. 9.2

[6] S. Ben-David, J. Blitzer, K. Crammer, and F. Pereira. Analysis of representations for domain adaptation. In *NIPS*, 2007. 2.2

[7] J. Bennett and S. Lanning. The Netflix prize. In *KDD Cup and Workshop*, 2007. 2.5, 8.1.1

[8] D. Berry and B. Fristedt. *Bandit Problems: Sequential Allocation of Experiments. 1985*. Chapman and Hall, London, 1985. 9

[9] J. Blitzer, R. Mcdonald, and F. Pereira. Domain adaptation with structural correspondence learning. In *In EMNLP*, 2006. 2.2

[10] Y. Boureau, F. Bach, Y. Le Cun, and J. Ponce. Learning mid-level features for recognition. In *CVPR*, 2010. 2.1

[11] W. Brendel and S. Todorovic. Activities as time series of human postures. In *ECCV*, 2010. 2.1

[12] R. S. C. Rudin, I. Daubechies. The dynamics of adaboost: Cyclic behavior and convergence of margins. *JMLR*, 5, 2004. 7.1.5, 7.4.2

[13] E. Candès and B. Recht. Exact matrix completion via convex optimization. *Commun. ACM*, 55(6):111–119, June 2012. 8.1.1

[14] L. Cao, Z. Liu, and T. Huang. Cross-dataset action detection. In *CVPR*, 2010. 2.2

[15] Carnegie Mellon University Graphics Lab. CMU graphics lab motion capture database, 2001. 3.1.5

[16] G. Carneiro. The automatic design of feature spaces for local image descriptors using an ensemble of non-linear feature extractors. In *CVPR*, 2010. 5.2

[17] R. Caruana. Multitask learning. In *Machine Learning*, pages 41–75, 1997. 2.3

[18] K. Chaloner and I. Verdinelli. Bayesian experimental design: A review. *Statistical Science*, 10(3):pp. 273–304, 1995. 8.2.1

[19] A. Chambolle and T. Pock. A first-order primal-dual algorithm for convex problems with applications to imaging. Preprint. 3.1.1

[20] C.-C. Chang and C.-J. Lin. LIBSVM – a library for support vector machines, 2001. 4.3.4

[21] O. Chapelle, P. Shivaswamy, S. Vadrevu, K. Weinberger, Y. Zhang, and B. Tseng. Multi-task learning for boosting with application to web search ranking. In *ACM SIGKDD*, 2010. 2.3, 7

[22] G. Chen, F. Wang, and C. Zhang. Collaborative filtering using orthogonal nonnegative matrix tri-factorization. *Inf. Process. Manage.*, 45(3):368–379, May 2009. 2.5

[23] Y. Chen, R. Parent, R. Machiraju, and J. Davis. Human activity recognition for synthesis. In *IEEE CVPR Workshop on Learning, Representation and Context for Human Sensing in Video*, 2006. 3.1.5

[24] H. Choset and P. Pignon. Coverage path planning: The Boustrophedon cellular decomposition. In *International Conference on Field and Service Robotics*, 1997. 9.1

[25] C. Cifuentes, M. Sturzel, F. Jurie, and G. Brostow. Motion models that only work sometimes. In *BMVC*, 2012. 2.4

[26] R. Collobert, F. Sinz, J. Weston, L. Bottou, and T. Joachims. Large scale transductive SVMs. *JMLR*, 7, 2006. 2.2

[27] W. Dai, Q. Yang, G. Xue, and Y. Yu. Boosting for transfer learning. In *ICML*, 2007. 2.1, 2.2

[28] DARPA. Mind's Eye Program. `http://www.darpa.mil/Our_Work/I2O/Programs/Minds_Eye.aspx`. 3.1.4

[29] H. Daumé III. Frustratingly easy domain adaptation. In *Proceedings of Association of Computational Linguistics*, 2007. 2.2

[30] 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. 2.5

[31] P. Dollar, V. Rabaud, G. Cottrell, and S. Belongie. Behavior recognition via sparse spatio-temporal features. In *VS-PETS*, 2005. 5

[32] L. Duan, D. Xu, I. Tsang, and J. Luo. Visual event recognition in videos by learning from web data. In *CVPR*, 2010. 2.2

[33] T. Evgeniou, C. A. Micchelli, and M. Pontil. Learning multiple tasks with kernel methods. *JMLR*, 6:615–637, 2005. 2.3

[34] J. Faddoul, B. Chidlovskii, F. Torre, and R. Gilleron. Boosting multi-task weak learners with applications to textual and social data. In *Machine Learning and Applications*, 2010. 2.3, 7

[35] R. Fergus, H. Bernal, Y. Weiss, and A. Torralba. Semantic label sharing for learning with many categories. In *ECCV*, 2010. 2.2, 10.2.1

[36] Y. Freund and R. Schapire. Experiments with a new boosting algorithm. In *ICML*, 1996. 7

[37] K. Goldberg, T. Roeder, D. Gupta, and C. Perkins. Eigentaste: A constant time collaborative filtering algorithm. *Inf. Retr.*, 4(2):133–151, July 2001. 2.2, 2.5

[38] R. Gopalan, R. Li, and R. Chellappa. Domain adaptation for object recognition: An unsupervised approach. In *ICCV*, 2011. 2.2

[39] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *JMLR*, 3:1157–1182, 2003. 2.1

[40] N. Jammalamadaka, A. Zisserman, and M. Eichner. Has my algorithm succeeded? an evaluator for human pose estimators. In *ECCV*, 2012. 2.4, 3.3.3, 6.5.2, 10.2.3

[41] Z. Kang, K. Grauman, and F. Sha. Learning with whom to share in multi-task feature learning. In *ICML*, 2011. 2.3

[42] A. Karatzoglou, X. Amatriain, L. Baltrunas, and N. Oliver. Multiverse recommendation: n-dimensional tensor factorization for context-aware collaborative filtering. In *Proceedings of the fourth ACM conference on Recommender systems*, RecSys '10, pages 79–86, 2010. 2.5

[43] R. H. Keshavan, A. Montanari, and S. Oh. Low-rank matrix completion with noisy observations: a quantitative comparison. In *Proceedings of the 47th annual Allerton conference on Communication, control, and computing*, Allerton'09, pages 1216–1222, 2009. 8.1.1

[44] R. H. Keshavan, A. Montanari, and S. Oh. Matrix completion from a few entries. *IEEE Trans. Inf. Theor.*, 56(6):2980–2998, June 2010. 8.1.1, 9.5

[45] A. Kläser, M. Marszałek, and C. Schmid. A spatio-temporal descriptor based on 3d-gradients. In *BMVC*, pages 995–1004, sep 2008. 3.1.1, 3.1.2, 7.3

[46] Y. Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *ACM KDD*, 2008. 9.2

[47] Y. Koren. The BellKor solution to the Netflix Grand Prize. `http://www.netflixprize.com/assets/GrandPrize2009_BPC_BellKor.pdf`, 2009. 2.5

[48] Y. Koren. Factor in the neighbors: Scalable and accurate collaborative filtering. *ACM Transactions Knowledge Discovery Data*, 4(1):1:1–1:24, 2010. 2.5, 6.2, 6.2, 9.2

[49] A. Kovashka and K. Grauman. Learning a hierarchy of discriminative space-time neighborhood features for human action recognition. In *CVPR*, 2010. 3.1.2, 7.3

[50] K. Lai and D. Fox. Object recognition in 3D point clouds using web data and domain adaptation. *International Journal of Robotics Research*, 29:1019–1037, 2010. 2.2

[51] T. Lai and H. Robbins. Asymptotically efficient adaptive allocation rules. *Advances in applied mathematics*, 6(1):4–22, 1985. 9.1

[52] I. Laptev and T. Lindeberg. Space-time interest points. In *ICCV*, 2003. 3.1.2, 7.3

[53] I. Laptev, M. Marszalek, C. Schmid, and B. Rozenfeld. Learning realistic human actions from movies. In *CVPR*, 2008. 2.1, 3.1.1, 4.2, 4.1, 5, 5.1

[54] 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. 2.2

[55] C. Li and K. Kitani. Pixel-level hand detection in ego-centric videos. In *Under submission to CVPR 2013*, 2013. 3.3

[56] H. Li, X. Liao, and L. Carin. Multi-task reinforcement learning in partially observable stochastic environments. *Journal of Machine Learning Research*, 2009. 9.1

[57] X. Lian, Z. Li, C. Wang, B. Lu, and L. Zhang. Probabilistic models for supervised dictionary learning. In *CVPR*, 2010. 2.1

[58] H. Liu, H. Motoda, and L. Yu. A selective sampling approach to active feature selection. *Artificial Intelligence*, 159:49–74, 2004. 1.2.2

[59] J. Liu, J. Luo, and M. Shah. Recognizing realistic actions from videos "in the wild". In *CVPR*, 2009. 3.1.2

[60] J. Loughrey and P. Cunningham. Overfitting in wrapper-based feature subset selection: The harder you try the worse it gets. In *Research and Development in Intelligent Systems XXI*, pages 33–43. Springer, 2005. 2.1

[61] O. Mac Aodha, G. Brostow, and M. Pollefeys. Segmenting video into classes of algorithm-suitability. In *CVPR*, 2010. 2.4

[62] O. Mac Aodha, A. Humayun, M. Pollefeys, and G. Brostow. Learning a confidence measure for optical flow. *PAMI*, 2012. 2.4, 6.5.2, 10.2.3

[63] D. Mackenzie and T. R. Balch. Making a clean sweep: Behavior based vacuuming, 1993. 9.1

[64] J. Mairal, F. Bach, J. Ponce, and G. Sapiro. Online learning for matrix factorization and sparse coding. *JMLR*, 11:19–60, 2010. 2.5

[65] J. Mairal, F. Bach, J. Ponce, G. Sapiro, and A. Zisserman. Supervised dictionary learning. In *NIPS*, 2008. 2.1

[66] S. Maji and J. Malik. Object detection using a max-margin Hough transform. In *CVPR*, 2009. 4.3.1

[67] P. Matikainen, M. Hebert, and R. Sukthankar. Trajectons: Action recognition through the motion analysis of tracked features. In *ICCV workshop on Video-oriented Object and Event Classification*, 2009. 4.1, 5

[68] P. Matikainen, M. Hebert, and R. Sukthankar. Representing pairwise spatial and temporal relations for action recognition. In *ECCV*, 2010. 2.1, 5, 5.1, 5.1, 5.4.2

[69] P. Matikainen, R. Sukthankar, and M. Hebert. Feature seeding for action recognition. In *ICCV*, 2011. 3.1.5

[70] V. B. Melas. *Functional Approach to Optimal Experimental Design (Lecture Notes in Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005. 8.2.1

[71] R. Messing, C. Pal, and H. Kautz. Activity recognition using the velocity histories of tracked keypoints. In *ICCV*, 2009. 4.1, 4.1, 5.1

[72] I. Mierswa and M. Wurst. Efficient case based feature construction for heterogeneous

learning tasks. In *ECML*, 2005. 2.2

[73] 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. 2.2

[74] J. Morra, T. Zhuowen, L. Apostolova, A. Green, A. Toga, and P. Thompson. Comparison of adaboost and support vector machines for detecting alzheimer's disease through automated hippocampal segmentation. *Medical Imaging, IEEE Transactions on*, 29(1):30 –43, jan. 2010. 7.1

[75] J. Niebles, H. Wang, and L. Fei-Fei. Unsupervised learning of human action categories using spatial-temporal words. *IJCV*, 79(3), 2008. 5

[76] G. Obozinski and B. Taskar. Multi-task feature selection. In *ICML Workshop on Structural Knowledge Transfer for Machine Learning*, 2006. 2.3

[77] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345 –1359, 2010. 2.3

[78] R. Parr and S. Russell. Reinforcement learning with hierarchies of machines. In *Advances in Neural Information Processing Systems 10*, pages 1043–1049. MIT Press, 1997. 9.1

[79] N. D. Phuong and T. M. Phuong. Collaborative filtering by multi-task learning. In *Research, Innovation and Vision for the Future, 2008. RIVF 2008. IEEE International Conference on*, pages 227 –232, july 2008. 2.5

[80] N. Pinto, J. DiCarlo, and D. Cox. How far can you get with a modern face recognition test set using only simple features? In *CVPR*, 2009. 2.1

[81] A. Rahimi and B. Recht. Random features for large-scale kernel machines. In *NIPS*, 2007. 5.2

[82] J. Rennie and N. Srebro. Fast maximum margin matrix factorization for collaborative prediction. In *Proceedings of the 22nd international conference on Machine learning*, ICML '05, pages 713–719, 2005. 2.5, 6.2

[83] H. Robbins. Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society*, 58(5):527–535, 1952. 9, 9.1

[84] B. Rosman and S. Ramamoorthy. A multitask representation using reusable local policy templates. *In Proc. AAAI Spring Symposium on Designing Intelligent Robots: Reintegrating AI*, 2012. 9.1

[85] U. Rückert and S. Kramer. Kernel-based inductive transfer. In *ECML*, 2008. 2.3

[86] S. Sadanand and J. Corso. Action bank: A high-level representation of activity in video. In *CVPR*, 2012. 3.1.3

[87] K. Saenko, B. Kulis, M. Fritz, and T. Darrell. Adapting visual category models to new domains. In *ECCV*, 2010. 2.2

[88] R. Salakhutdinov and A. Mnih. Bayesian probabilistic matrix factorization using markov chain monte carlo. In *Proceedings of the 25th international conference on Machine learning*, ICML '08, pages 880–887, 2008. 2.5

[89] B. Sallans, G. E. Hinton, and S. Mahadevan. Reinforcement learning with factored states

and actions. *Journal of Machine Learning Research*, 5:1063–1088, 2004. 9.1

[90] S. Satkin, J. Lin, and M. Hebert. Data-driven scene understanding from 3D models. In *BMVC*, 2012. 3.2, 3.2.3, 6.5.2

[91] J. Schmidhuber. Adaptive confidence and adaptive curiosity. Technical Report FKI-149-91, Institut fur Informatik, Technische Universitat Munchen, 1991. 9.1

[92] C. Schuldt, I. Laptev, and B. Caputo. Recognizing human actions: a local SVM approach. In *ICPR*, 2004. 2.2, 5

[93] J. Shawe-Taylor and N. Cristianini. *Support Vector Machines and other kernel-based learning methods*. Cambridge University Press, 2000. 5.2

[94] L. Shen and L. Bai. Adaboost gabor feature selection for classification. In *Proceedings of the Image and Vision Computing Conference, New Zealand*, 2004. 7.1

[95] J. Shotton, A. Fitzgibbon, M. Cook, and A. Blake. Real-time human pose recognition in parts from single depth images. In *CVPR*, 2011. 3.1.5

[96] N. Srebro and T. Jaakkola. Weighted low-rank approximations. In *In 20th International Conference on Machine Learning*, pages 720–727. AAAI Press, 2003. 2.5, 6.2

[97] D. Stavens and S. Thrun. Unsupervised learning of invariant features using video. In *CVPR*, 2010. 2.1

[98] Z. Szabó, B. Póczos, and A. Lőrincz. Collaborative filtering via group-structured dictionary learning. In *Proceedings of the 10th international conference on Latent Variable Analysis and Signal Separation*, 2012. 2.5

[99] F. Tanaka and M. Yamamura. Multitask reinforcement learning on the distribution of MDPs. In *Proceedings of IEEE International Symposium on Computational Intelligence in Robotics and Automation*, volume 3, pages 1108 – 1113, july 2003. 9.1

[100] A. Torralba and A. Efros. Unbiased look at dataset bias. In *CVPR*, 2011. 3.1.5

[101] M. Turk and A. Pentland. Face recognition using eigenfaces. In *CVPR*, 1991. 6.3.2

[102] University of Central Florida. UCF50 action recognition dataset. `http://server.cs.ucf.edu/~vision/data.html#UCF50`, 2011. 3.1.2, 3.1.5, 7.3

[103] X. Wang, C. Zhang, and Z. Zhang. Boosted multi-task learning for face verification with applications to web image and video search. In *CVPR*, 2009. 2.1, 2.3, 7

[104] A. Wilson, A. Fern, S. Ray, and P. Tadepalli. Multi-task reinforcement learning: A hierarchical bayesian approach. In *Proceedings of the International Conference on Machine Learning*, 2007. 9.1

[105] T. Windeatt and K. Dias. Feature ranking ensembles for facial action unit classification. In *Proceedings Artificial Neural Networks in Pattern Recognition*, 2008. 2.1

[106] S. Wong and B. MacDonald. A topological coverage algorithm for mobile robots. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 2, pages 1685 – 1690 vol.2, oct. 2003. 9.1

[107] M. Wu. Collaborative filtering via ensembles of matrix factorization. In *KDDCup 2007*, pages 43–47, 2007. 2.5, 8.1.1

[108] J. Xiao, J. Hays, K. Ehinger, A. Oliva, and A. Torralba. Sun database: Large-scale scene recognition from abbey to zoo. In *CVPR*, 2010. 3.2.2

[109] L. Xiong, X. Chen, T. K. Huang, J. Schneider, and J. G. Carbonell. Temporal collaborative filtering with bayesian probabilistic tensor factorization. In *SIAM Data Mining 2010 (SDM 10)*, 2010. 2.5

[110] L. Yang, R. Jin, C. Pantofaru, and R. Sukthankar. Discriminative cluster refinement: Improving object category recognition given limited training data. In *CVPR*, 2007. 2.1

[111] J. Yuan, Z. Liu, and Y. Wu. Discriminative subvolume search for efficient action detection. In *CVPR*, 2009. 3.1.5, 5.4

[112] M. Zhou and H. Wei. Face verification using gabor wavelets and adaboost. In *ICPR*, 2006. 7.1

[113] X. Zhou, X. Zhuang, M. Liu, H. Tang, M. Hasegawa-Johnson, and T. Huang. HMM-based acoustic event detection with adaboost feature selection. *Multimodal Technologies for Perception of Humans*, 4625:345–353, 2008. 7.1