A Task-Oriented Approach for Cost-sensitive Recognition Supplementary Material

Roozbeh Mottaghi[1] Hannaneh Hajishirzi[2] Ali Farhadi[1,2]
[1]Allen Institute for Artificial Intelligence
[2]University of Washington

1. Order of activation of features

Figure 1 shows the order of activation of features for each known SHUT. Suppose features $\{A, B\}$ are selected for the current value of λ. Suppose feature C replaces $\{A, B\}$ if we decrease λ. The order of activation of features will be (A, B, C) since C gets activated after A and B.

2. Zero-shot learning with different number of clusters

Figure 2 shows the results of zero-shot learning for different number of clusters. The rightmost column corresponds to the case that each cluster includes only one SHUT. The result for that case is worse compared to some other cases that are based on grouped SHUTs. Therefore, clustering is effective in improving the performance for unseen SHUTs.

3. Details of computing features

This section describes the details of computing features. We use RGB-D data in this paper, therefore, our features are defined for regions that span a volume in 3D, and correspond to a set of pixels in the 2D image. We require regions that cover the entire objects and do not overlap more than one object as much as possible. We employ the region generation method of [8]. For our experiments, we use the regions generated in the 5^{th} level of the hierarchical segmentation that [8] provides. Regions in that level provide a reasonable overlap with object and non-object instances. We obtain 95 regions per image on average by the above method. The features are described below:

Height: This feature measures the height of the region. The intuition is that for some tasks (e.g., *walking*) the height of the region plays an important role and it is impossible to walk on surfaces of more than certain height.

We rotate the depth map such that the normal of the floor points upward using the method described in [8]. The height of each pixel is basically the y coordinate of that pixel in the rotated depth map. To obtain the feature for each region, we compute the histogram of heights for the con-

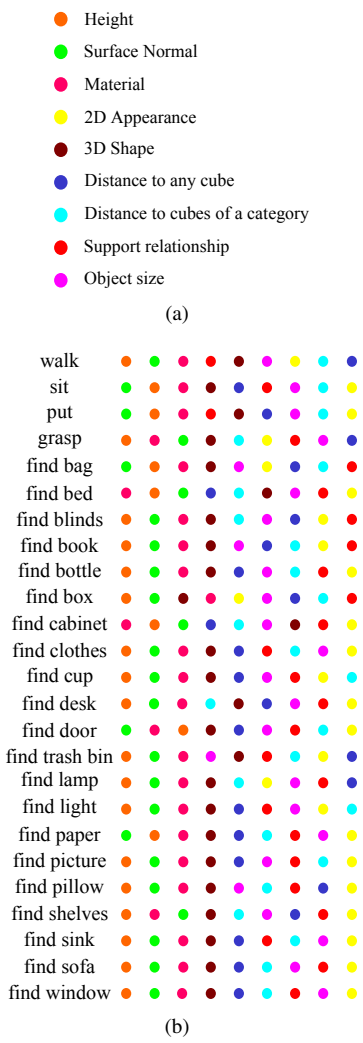


Figure 1: (a) The corresponding color of each feature is shown. (b) Order of activation of the features (from left to right).

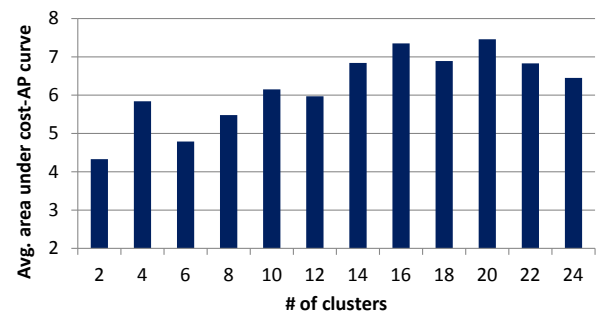


Figure 2: Results of zero-shot learning using different number of clusters. The results for 20 clusters are shown in Table 1 (bottom) of the paper.

stituent pixels of a region. This feature has 20 dimensions (the interval between the maximum and minimum height in the dataset is divided into 20 bins).

Surface Normals: This feature represents the surface normals of a region. Similar to above, the depth map is rotated. Then, we compute the histogram of surface normals for the pixels of a region. The histogram is computed for x, y, and z coordinates separately. This feature has 30 dimensions, where we consider 10 bins for each coordinate.

Material Attributes: Another useful feature for our tasks is the material of the regions. For instance, it is unlikely that a suitable surface for *sitting* surface is made of *glass*. We used the annotations of [11], which provide additional attribute annotations: *wood*, *painted*, *cotton*, *glass*, *glossy*, *plastic*, *shiny*, *textured*.

We train a linear multi-class SVM on a 1024 dimensional histogram of SIFT codebooks, where the output label is one of the eight material attributes that we consider. The feature is the confidence for each attribute so it has 8 dimensions.

2D Appearance: To capture the 2D appearance or texture of the regions, we use kernel descriptors (KDES) [1]. Basically, KDES converts pixel-level similarity functions (kernels) to descriptors for a patch. [7] uses 6 types of kernels: gradient, color, local binary pattern, depth gradient, surface normal, and KPCA/self-similarity. Kernel descriptors are combined over regions using Efficient Match Kernels (EMK) [2]. We ignore the 3D kernels to compute this feature.

3D Shape: We use 3D shape as one of our cues since the tasks that humans perform happen in 3D environments, and 2D images alone are ambiguous in that the entire 3D structure of the scene is projected onto a 2D image.

We use the implementation of [9] for computing the 3D shape features. Their feature is comprised of four different 3D cues. The features are computed on cubic cells, which are generated by dividing the 3D scene into cubic cells of size 0.1.

The first cue is the *point density*, which shows the number of points in each voxel inside a cell (the cells are divided into $6 \times 6 \times 6$ voxels). A histogram represents the number of points in each voxel. In addition to the first order statistics, the second order statistics (count difference) are also considered. For more details, refer to [9].

The second cue is the distribution of points inside each voxel. The point cloud distribution inside each voxel is represented by *scatter-ness*, *linear-ness*, and *surface-ness*, which are computed based on the principal components of the point cloud.

The third cue is 3D normals, which is similar to our surface normal feature. The fourth cue captures the global shape statistics and it is based on Truncated Signed Distance Function (TSDF). For each voxel, the distance between its center and the nearest object point on the line of sight of the camera is used to compute this cue.

All of these cues are represented as a histogram for a specific cell. To compute the feature for our regions, we check which cells overlap the regions and compute the average of the histograms corresponding to the cells that overlap with the region.

Distance to *any* Object: The intuition for this feature is that for some tasks, we might need to know only the distance to the surrounding objects and we do not need to reason about the actual appearance of the surrounding objects.

To compute this feature, we rely on a set of object cuboid hypotheses that are generated by the method of [6]. First, a set of 3D regions are generated from the scene point cloud in a bottom-up fashion (extension of CPMC [3] to 3D). We use the setting that generates 15 cuboids per image as it produces a reasonable recall for the dataset that we use for our experiments. Our feature has 6 dimensions, where the first three dimensions represent the distance between a region and the closest cuboid visible in the image. The distance corresponds to the distance of the closest points of the cuboid and the region in 3D. We consider the distance in x, y, and z coordinates separately. The second three dimensions correspond to the average of the distances between the region and the 15 cuboids extracted for an image. If a region overlaps with a cube, the distance will be zero. Figure 3 (in the paper) illustrates the cubes and their distance to a region.

Distance to Instances of a Particular Category: In computing the above feature, we ignored the category of the object, but in some cases the category of the surrounding objects (contextual information) is very informative for a task. For instance, there is a high chance that a surface next to a *cup* is a suitable surface for *putting* since it probably supports the cup, and the cup is in a stable condition. Hence, it might be a stable surface for other objects too.

For this feature we again rely on the method of [6], but we use their object classification results. Their method de-

termines the category of each cuboid (among the 21 categories they consider and background). We consider 6 dimensions for each category of interest (+background). Hence, for K object categories, we have have $6(K+1)$ entries in the feature vector. Similar to above, for each category, we use the distance of the region to the closest cuboid of a particular category in x, y, and z coordinates. We also use the average distance to all of the cuboids of a category (similar to the average used for computing the above feature). If a category does not exist in an image, we zero out all of the 6 entries corresponding to that category and set the bias term to 1. We have K bias terms in the feature vector so the total size of the feature vector is $6(K+1)+K$. Note that we rotate the scene so that the y direction points upward and we get a consistent set of distances across images. We use the categories that [6] predicts so $K = 21$ in our experiments.

Support Relations: Support relationships are another important cue for task-based reasoning. For example, if a surface is a *supporter* for an object, there is a high chance that it is a good candidate for *putting*.

To compute this feature, we use the method of [8] that estimates support relationships between pairs of regions in a scene. Their assumption is that a region can have one of the following states: (1) supported from behind, (2) supported from below, (3) supported by a hidden object, (4) ground (not supported by other regions). [8] predicts the support state based on a set of features that are defined on pairs of regions.

We compute our support feature based on the output of the prediction made by [8]. Our feature has the following form: $\mathbf{f} = [\mathbf{f}_{bh}, \mathbf{f}_{bl}, b_{bh}, b_{bl}, b_g, b_h]$, where the first two terms correspond to the actual features and the last four terms are bias terms. If the predicted state for region i is state (1) (defined above), we find region j that supports i (according to the prediction). We use the feature that [8] defines on the pair $i-j$ as $\mathbf{f}_{bh}$. In this case, $b_{bl} = 1$, and the rest of the terms will be zero. Similarly, for predicted state (2), only $\mathbf{f}_{bl}$ and b_{bh} are set, and the rest of the terms will be 0. If the predicted state is (3) or (4), we set b_{bl} and b_{bh} and either b_g or b_h (depending on the ground or hidden prediction) to 1. $\mathbf{f}_{bh}$ and $\mathbf{f}_{bl}$ will be zero in this case.

Object Size: We hypothesize that object size is another important cue for task-based recognition. For instance, a large object (e.g., *bed*) cannot be *grasped*. Our regions sometimes cover part of objects so their size is not a good indicative of the object size. Therefore, we rely on the object cuboids generated by [6] (used above). For each region, we find the cuboid that has the largest overlap with the region (the overlap is defined as the size of the intersection of the region and the cuboid in 3D divided by the region size). The feature for each region is the cuboid volume, area of the largest surface, and area of the smallest surface of the cuboid, hence the feature has 3 dimensions. We use the setting of [6] that generates 15 cuboids per image.

4. Tasks

The results of Figure 7 in the paper are based on the following 15 tasks:

1. Put the cup on the shelf.
2. Sit on the sofa.
3. Put the book on the desk.
4. Walk to the door.
5. Put pillow on the bed.
6. Sit on the desk.
7. Walk away from the window.
8. Grasp the bottle.
9. Put the cup in the sink.
10. Walk towards the window with blinds.
11. Grasp the bag.
12. Grasp the papers and walk towards the trash bin.
13. Put the box next to the cabinet.
14. Put the papers in the trash bin.
15. Walk to find my clothes.

5. Further implementation details

We consider a rough estimate for the feature costs since the feature computation time depends on the complexity of the image, type of objects present in a scene, etc. The costs that we considered for computing features for each image are as follows (they are just relative costs and do not have a specific unit): Height: 1, Surface normal: 1, Material attributes: 2, 2D appearance: 20, 3D shape: 15, Distance to any object: 10, Distance to instances of a particular category: 30, Support relations: 20, and Object size: 10.

For the baseline methods in Table 1 of the paper [5, 10], we used their publicly available implementation. For [5], we used KNN-Euclidean setting since it produced the best results for that method. For the linear SVM with L1 regularization we used the implementation of [4].

To compute our curves we used 20 different values of λ between 10^{-7} and 0.05.

6. Curves for known SHUTs in Table 1

In Figure 3, we provide the curves corresponding to four main methods mentioned in Table 1 (top) in the paper.

References

[1] L. Bo, X. Ren, and D. Fox. Kernel descriptors for visual recognition. In *NIPS*, 2010. 2

[2] L. Bo and C. Sminchisescu. Efficient match kernel between sets of features for visual recognition. In *NIPS*, 2009. 2

[3] J. Carreira and C. Sminchisescu. Segmentation using constrained parametric min-cuts. *TPAMI*, 2012. 2

[4] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. Liblinear: A library for large linear classification. *JMLR*, 2008. 3

[5] S. Karayev, M. Fritz, and T. Darrell. Anytime recognition of objects and scenes. In *CVPR*, 2014. 3

[6] D. Lin, S. Fidler, and R. Urtasun. Holistic scene understanding for 3d object detection with rgbd cameras. In *ICCV*, 2013. 2, 3

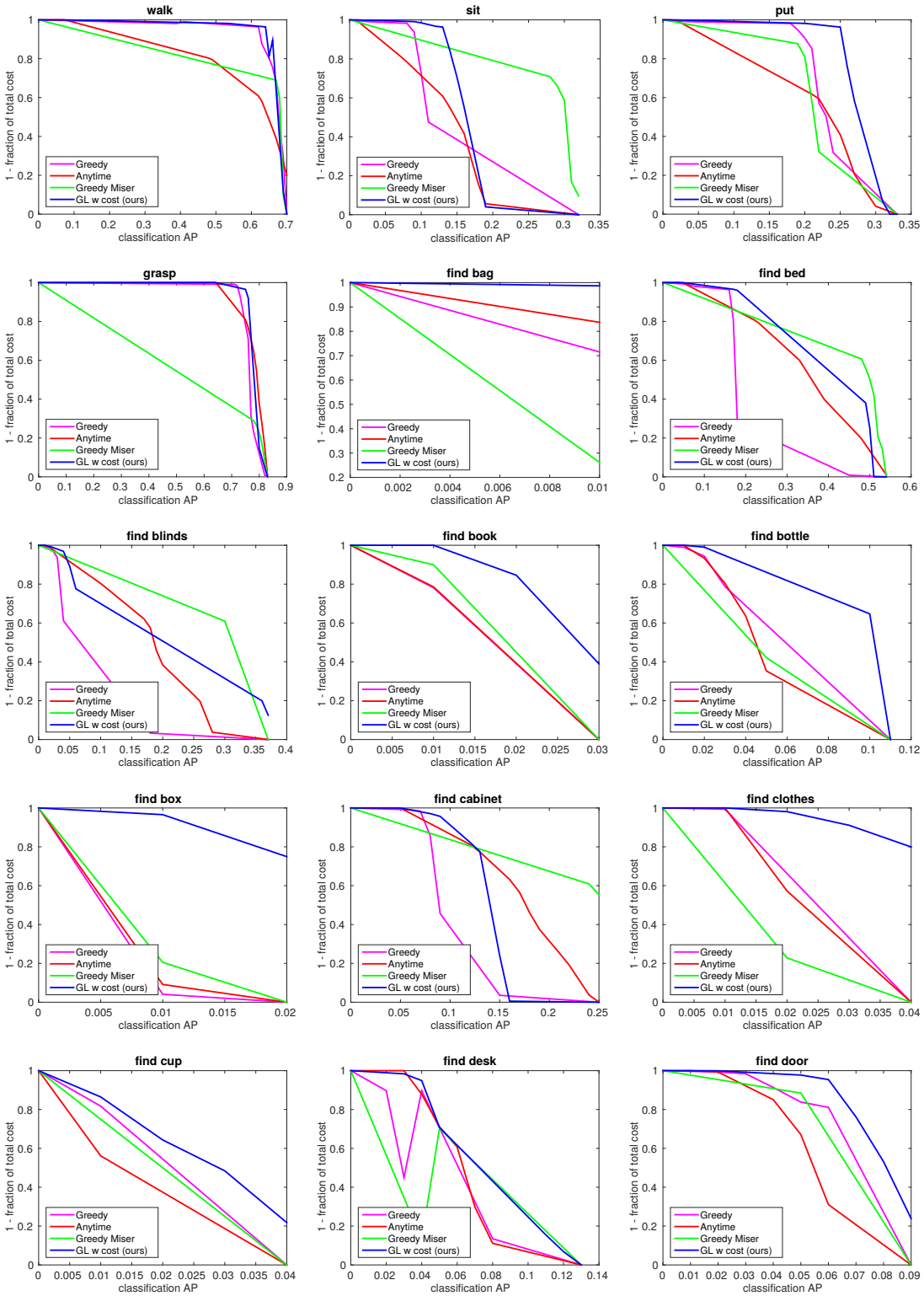
[7] X. Ren, L. Bo, and D. Fox. Rgb-(d) scene labeling: Features and algorithms. In *CVPR*, 2012. 2

[8] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus. Indoor segmentation and support inference from rgbd images. In *ECCV*, 2012. 1, 3

[9] S. Song and J. Xiao. Sliding shapes for 3d object detection in depth images. In *ECCV*, 2014. 2

[10] Z. Xu, K. Weinberger, and O. Chapelle. The greedy miser: Learning under test-time budgets. In *ICML*, 2012. 3

[11] S. Zheng, M.-M. Cheng, J. Warrell, P. Sturgess, V. Vineet, C. Rother, and P. H. Torr. Dense semantic image segmentation with objects and attributes. In *CVPR*, 2014. 2


walk
sit
put
grasp
find bag
find bed
find blinds
find book
find bottle
find box
find cabinet
find clothes
find cup
find desk
find door
1 - fraction of total cost
classification AP
Greedy
Anytime
Greedy Miser
GL w cost (ours)

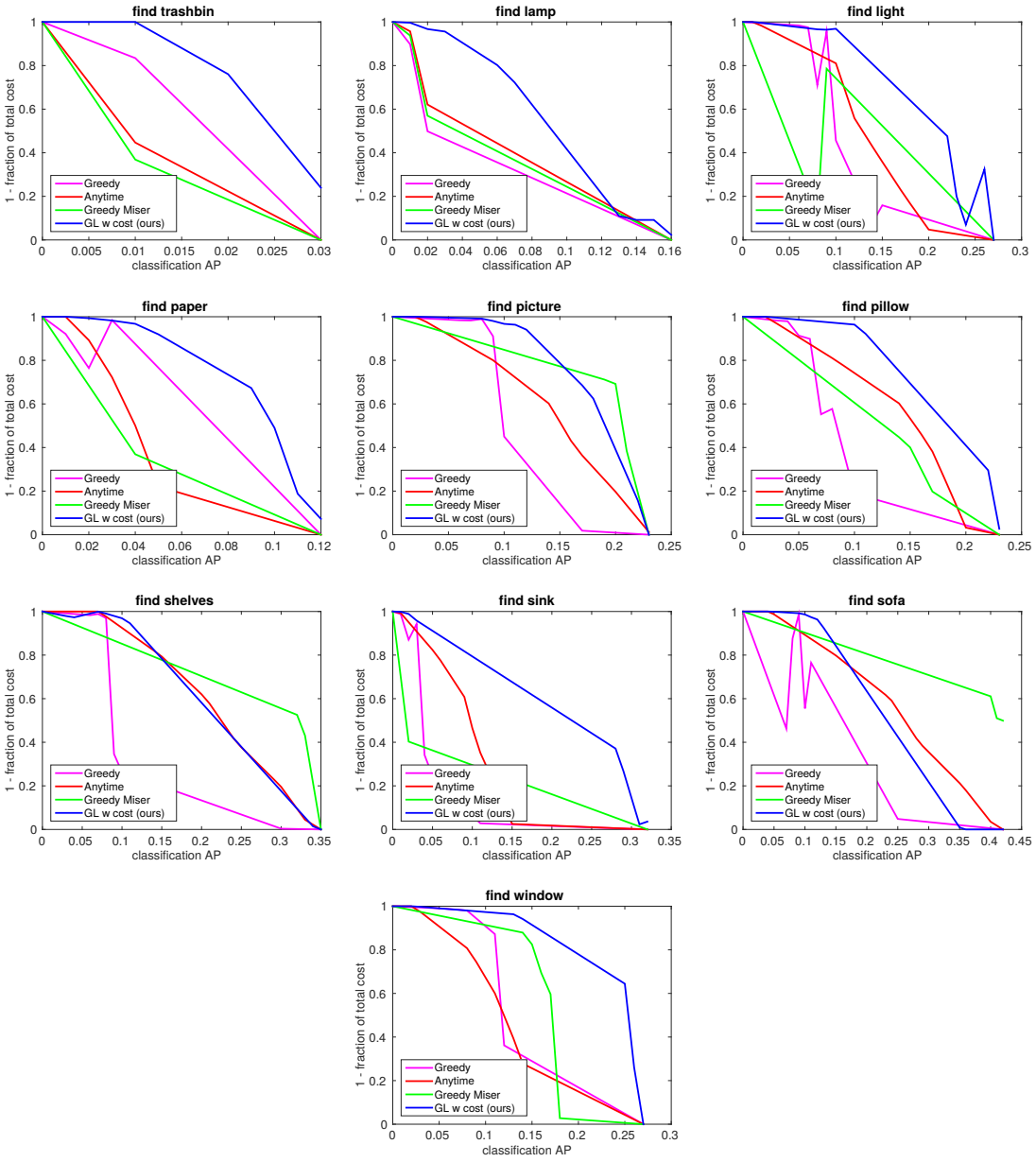


Figure 3: Classification AP vs (1-fraction of total cost) plots.