6.867: Machine Learning

Applying Machine Learning Techniques to Improve the Object Detector Used by LIS

Ariel Anders Sanja Popovic

Massachusetts Institute of Technology
Fall 2012

Contents

1 Introduction

Our goal is to use different machine learning techniques to improve our research group's object detector designed by Jared Glover. The input for the object detector consists of two RGBD point-clouds. One represents the object model and the other is the scene in which we are searching for the given object. The output is a hypothesis of the object location in the scene. Each hypothesis is a six degree of freedom position relative to the viewpoint of the scene.

For each hypothesis there are four log likelihood scores:

ϕ_1 : Distances between pairs of corresponding points
ϕ_2 : Shape similarity between pairs of corresponding points
ϕ_3 : Color similarity between pairs of corresponding points
ϕ_4 : Number of matching points within the scene

Currently, the overall score S for the hypothesis is the sum of the scores:

$$S = \sum_{n=1}^{4} \phi_n \tag{1}$$

In order to obtain a hypothesis, the algorithm goes through rounds of feature matching. In each round, it selects a feature on the model, finds its correspondent feature in the scene and establishes numerous potential poses based on this correspondence. It scores the poses as described above and it keeps only the top 10% for further refinement. Using the remaining hypotheses, the object detector will establish additional correspondences to align the object model with the scene.

The datasets we are using have the ground truth of the location of the target objects within the scene. Using the ground truth, we can determine if a hypothesis is good or not by using the following formula:

$$classification(\hat{X}) = \begin{cases} \text{good}, & ||\Delta_x|| < 0.05 \wedge ||\Delta_q|| < \frac{\pi}{8} \\ \text{bad}, & \text{otherwise} \end{cases}$$

where Δ_x is the normalized translation discrepancy from the true position and Δ_q is the normalized rotational discrepancy from the true position.

We can rate relative hypothesis by computing a single scalar distance metric:

$$D = \frac{1}{0.05} \cdot \Delta_x + \frac{8}{\pi} \cdot \Delta_q = 20 \cdot \Delta_x + \frac{8}{\pi} \cdot \Delta_q \tag{2}$$

While running the object detector on for example the `409Bottle`, an object model in our dataset, we found 36 good hypothesis were found out of 1000 generated hypothesis. However, after sorting and pruning the top 10% only 31 good hypothesis remained. Our goal is to find a way to keep the 5 poorly ranked hypothesis by changing the score function to

$$S = \sum_{n=1}^{4} \omega\phi_n \tag{3}$$

where ω is a 4-element column vector that we determine using machine learning techniques.

This paper is structured as follows. It starts with the first approach to solve this problem using ordinary regression to establish a baseline for our experiments (section 2). Section 3 discusses the

use of the ordinal regression to find ω . Finally, we discuss our third approach using P-norm push in section 4. Section 5 summarizes the results of the three approaches and we conclude by a detailed analysis of our experimental results in section 6.

2 Regression

We chose regression as a representative technique we learned in class. Our idea was to establish a correlation between the scores ϕ and discrepancy D between the estimated pose and true pose so that high-scoring estimations have small discrepancy from the true pose. In other words, we wanted the weighted score to be inversely proportional to the distance ($\frac{1}{D} \propto \omega\phi$).

We examined three different functions categories to fit our data with: $f(x) = kx+n$, $f(x) = e^{-x}$, and $f(x) = e^{-x^2}$. Figure 1 shows graphs of each of these functions.

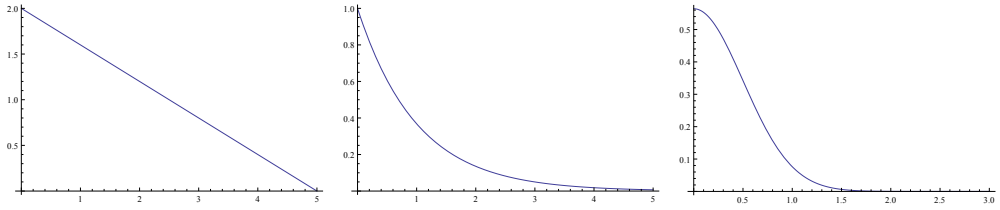
From the linear graph we can see that the function does not decrease very quickly which will result in relatively high scores being given to relatively bad hypotheses. On the other hand, the exponential function decreases too quickly and will penalize relatively good hypotheses.

Thus, we settled on the third function $f(x) = e^{-x^2}$ (the form of a Gaussian with mean 0) because it has a peak that would be tolerant to some noise. Additionally, by changing variance, we can control the width of the peak.

In order to scale our scores with the Gaussian, we augmented the score vector ϕ with 1 at the beginning and added w_0 to the weight vector.

2.1 Objective Function

$$E = \frac{\lambda}{2}||\omega|| + \frac{1}{2}\left(\frac{1}{\sqrt{2\pi}}e^{-\frac{D^2}{2}} - e^{\omega\phi+\omega_0}\right)^2 \tag{4}$$



(a) Linear (b) Exponential (c) Gaussian

Figure 1: Score versus Distance Discrepancy

3 Ordinal regression

In this section, we will describe the method of using a pairwise ranking approach for Ordinal Regression introduced by Ralf Herbrich et al.[**?**] and Thorsten Joachims[**?**]. This pairwise classification approach transforms each pair of training data points with unequal target values into a new data set D_{new} by the following transformation:

$$D_{new} := \{(x,t)|x = (\phi^{(1)} - \phi^{(2)}),\ t = sgn(D^{(1)} - D^{(2)}),\ \text{s.t.} D^{(1)} \neq D^{(2)}\} \tag{5}$$

where sgn is the sign function.

Herbrich et al. proposed that finding ω for the ranking function $f(\phi) = \omega\phi$ can be done by maximizing the margin $sgn(D^{(1)} - D^{(2)}) \cdot (\omega\phi^{(1)} - \omega\phi^{(2)})$; thus, finding ω can be formalized as the following Support Vector Machine problem [**?**]:

$$w^* = \sum_i \alpha_i t_i (\phi_i^{(1)} - \phi_i^{(2)}) = \sum_i \alpha_i t_i x_i \tag{6}$$

$$\alpha^* = \arg\max_\alpha \left[\sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j t_i t_j ((\phi^{(1)} - \phi^{(2)}) \cdot (\phi^{(1)} - \phi^{(2)})) \right] \tag{7}$$

$$= \arg\max_\alpha [\sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j t_i t_j x_i \cdot x_j] \text{ where } C > \alpha > 0, \text{ and } \alpha \cdot t = 0 \tag{8}$$

The quadratic scaling blow-up made this algorithm extremely inefficient and we were unable to obtain any relevant results. However, Joachim proposed a slightly different method for improving retrieval quality of search engines using data from different queries[**?**]. Joachim's modification is to only compute the pairwise differences for data from the same group (or query). We extended this approach to our algorithm by grouping points from the same scene and object model. For M total groups, the new SVM problem is formulated as described below.
Note that this is the equivalent of equation 5, except that it is for grouped points.

$$D_{new} := \{(x_m, t_m)|x = (\phi_i^{(1)} - \phi_i^{(2)}), t = sgn(D_i^{(1)} - D_i^{(2)}), \forall D_i^{(1)} \neq D_i^{(2)} \quad i = 1, \ldots, M\} \tag{9}$$

The equivalent SVM problem is [**?**]:

$$min_{w,\xi} \frac{1}{2}||w||^2 + C \sum_{i=1}^{M} \xi_i,\ \text{s.t.} y_i w \cdot (\phi_i^{(1)} - \phi_i^{(2)}) \geq 1 - \xi_i, \quad i = 1, \ldots, M \tag{10}$$

This is equivalent to minimizing the regularized hinge loss function with $\lambda = \frac{1}{2C}$ [**?**].

$$min_w \sum_{i=1}^{M} [1 - y_i w \cdot (\phi_i^{(1)} - \phi_i^{(2)})]_+ + \lambda ||w||^2 \tag{11}$$

where $[x]_+$ denotes the function $\max(x, 0)$ for $i = 1, \ldots, M$.

One of the difficulties with the pairwise approach (without grouping) is that it scales the training data quadratically making it intractable to solve with the size of our data sets. However, by using

the pairwise grouping between objects from only the same group we were able to greatly decrease the size of D_{new} and were able to achieve better results using this method. Unfortunately, our Matlab implementation was still unable to solve large training data sets and took an extremely long time to converge to a solution even on smaller sized data-sets. To verify the correctness of our matlab implementation we tested against SVM^{rank}[**?**] by Joachims, a C implementation of SVM ranking. Since this C implementation has much quicker running time and is able to handle larger data sets, we chose to use their software for finding w^*. The following section outlines how we verified that our ordinal regression function did produce consistent results with SVM^{rank} for smaller data sets.

3.1 Validity of Our Implementation of Ordinal Regression

First, we verified our implementation with SVM^{rank} using their example data set and verified that our simulation had equivalent results. Their example file had 12 training data points and 4 test points. We trained w on the 12 training points and then used it to rank the 4 test points which had equivalent results. In addition, we used the same weights to rank the training data and ran the training data through the SVM^{rank} classifier for extra confirmation.

Second, we verified the validity of our ordinal regression by generating small data sets from our object detector data. Once again, we saw our results produced consistent rankings with SVM^{rank}. However, we did find a very small discrepancy within the normalized weight vectors found by our implementation (us) versus SVM^{Rank} shown in the table below.

$\hat{\omega_{us}}$	-0.7256	0.3043	0.6170	0.0178
$\hat{\omega_{SVM^{rank}}}$	-0.7255	0.3042	0.6171	0.0178

Normalized Weight Vector Validation Comparison $\left(\frac{w}{||w||}\right)$ for a small dataset.

Since we were able to exhibit consistent results with SVM^{rank} on different data sets and obtained almost equivalent resultant weight vectors, we are confident that our implementation is consistent with SVM^{rank}.

4 P-norm push

The last method we tried using was P-norm push by Cynthia Rudin [**?**]. P-norm push seemed like a good approach for our problem since the algorithm spends most of its efforts to rank the top of the list correctly. This implies pushing the good hypotheses closer to the top and doing a mediocre ranking on the rest of the list. Through parameter p, we can decide where we want to focus - high p means that all efforts will be spent on the very top of the list, while lower p spreads our efforts thinner working on a larger portion of the list. The algorithm falls into the category of boosting algorithms, in particular, it is a generalization of RankBoost [**?**].

4.1 Algorithm description

The algorithm takes two sets of points as an input, the positive ones, $\{\mathbf{x}_i\}_{i=1,\dots,I}$ and the negative ones, $\{\hat{\mathbf{x}}_k\}_{k=1,\dots,K}$. Throughout this section, we will use i in the context of positive points and k in the context of negative ones. The goal is to find a scoring function f where we want $f(\mathbf{x}_i) > f(\hat{\mathbf{x}}_i)$

for every pair (i,k). In order to minimize the pairs where $f(\mathbf{x}_i) < f(\hat{\mathbf{x}}_i)$, we will establish the following objective function:

$$R_{g,l}(f) = \sum_{k=1}^{K} g\left(\sum_{i=1}^{I} l(f(\mathbf{x_i}) - f(\hat{\mathbf{x_k}})) \right)$$

In this formula, l is the loss function and g is the cost function. For example, l can be a 0-1 loss, where we only care about the number of pairs where $f(\mathbf{x}_i) < f(\hat{\mathbf{x}}_i)$. In training phase, it is more common for the loss to be a smoother function, like $l(r) = e^{-p}$. The common choice of a cost function $g(r)$ is $g(r) = e^r$ or $g(r) = r^p$.

The function $f(\mathbf{x})$ is of a form $f(\mathbf{x}) = \sum_j \lambda_j h_j(\mathbf{x})$ where h_j is a weak classifier that returns a value $[0, 1]$ and λs are weights of the classifiers. In order to obtain h_j and λs, we perform t_{max} rounds of boosting. While performing boosting, we observe the difference between output values of two classifiers on a pair of positive and negative points. We will use $M_{i,k,j} = h_j(\mathbf{x}_i) - h_j(\hat{\mathbf{x}}_k)$ to denote the difference between the outputs of j-th classifier on i-th positive point and k-th negative point. We will use $d_{t,i,k}$ as the weight of pair $(\mathbf{x}_i, \hat{\mathbf{x}}_k)$ in t-th round of boosting (all $d_{t,i,k}$ are initialized to $1/IK$). To gain a better intuition of the algorithm, let us observe $M_{i,k,j}$. For simplicity, let us assume that the classifiers are binary. In that case $M_{i,k,j}$ is 1 if both points are classified correctly, -1 if both are classified wrong an 0 if one is correct and one is incorrect.

Let us now observe t-th round of the boosting portion of the algorithm. For this part of the analysis, we will assume our objective function is given as

$$R_{g,l}(\lambda_t) = \sum_{k=1}^{K} \Big(\sum_{i=1}^{I} e^{(-(\sum_j \lambda_j h_j(\mathbf{x}_i) - \sum_j \lambda_j h_j(\hat{\mathbf{x}}_k))} \Big)^p$$

1. We are searching for j_t such that

$$\sum_{i,k} d_{t,i,k} \left(\left(\sum_{i'} d_{t,i'k} \right)^{p-1} M_{i,k,j} \right)$$

   is maximized. Due to the nature of M, this step will choose a classifier that gets most of the important pairs correctly. As we can see, increasing p means that negative points that are closer to the top (i.e. that might be ranked above many positive points) are heavily penalized.

2. Let us now find α_t that minimizes the objective $R_{g,l}(\lambda_t + \alpha \mathbf{e}_{jt})$ in the j_t-th direction (note that $\mathbf{e}_{jt}$ is a row-vector that is zero everywhere except in the j_t-th column.

3. We will use α_t for the new value of weights. We will set $\lambda_{t+1} = \lambda_t + \alpha \mathbf{e}_{jt}$.

4. We will now update and normalize the weights on all positive-negative pairs:
   $z_t = \sum_{i,k} d_{t,i,k} e^{-\alpha_t M_{i,k,j_t}}$
   $d_{t+1,i,k} = d_{t,i,k} e^{-\alpha_t M_{i,k,j_t}} / z_t$ for every pair (i,k)

Finally, as an output, we get $\lambda_{t_{max}}$.

4.2 Difficulties with Implementation

Before we ran the P-norm push on our data, we wanted to ensure the correctness of our implementation. We designed a few very small datasets where we could check the correctness of the output by hand as a sanity check. Once we passed those checks, we wanted to replicate the results from Rudin's paper since the datasets were publicly available. However, this turned out to be an unsurmountable obstacle. Because the complexity of the algorithm grows quickly with the number of points and each round of boosting includes expensive function minimizations (to find α_t), it was intractable for us to test the algorithm on larger datasets. Even for smaller datasets, each run took about 2 hours which made potential debugging impossible. Additionally, we noticed some special cases that need to be handled separately, but the paper was very vague on how to work around those issues. Namely, the issues were about obtaining α_t. Once we set the partial derivative of $R_{g,l}(\lambda_t + \alpha \mathbf{e}_{jt})$ to 0 with respect to α_t we get the following:

$$0 = \sum_{k=1}^{K} \left[\left(\sum_{i=1}^{I} d_{t,i,k} e^{-\alpha_t M_{i,k,j_t}} \right)^{p-1} \left(\sum_{i=1}^{I} M_{i,k,j_t} d_{t,i,k} e^{-\alpha_t M_{i,k,j_t}} \right) \right].$$

Again, for simplicity, let us assume binary classifiers. Observing the first inner sum shows this factor is always non-negative. The second factor's sign depends on the M_{i,k,j_t} values. In case the observed classifier is a perfect separator all M_{i,k,j_t}'s are non-negative and the only way to drive this derivative to 0 is to set $\alpha = +\infty$ which consequently drives λs to infinity. If the separator is perfect, this result is reasonable and expected because we want to put all the weight on it. However, the same issue happens when the separator is perfect on only one side. In that case, all M_{i,k,j_t}s are 1 or 0. This again brings α to $\pm\infty$ which may not be the desired behavior. After trying many hacks to work around this issue, we decided to contact the author. Professor Rudin suggested using regularization on α which did fix the infinity issue, but we still could not obtain the results from the paper. Of course, we did not aim to replicate the exact values, just the general trends, but we failed at that task as well. We did notice, however, that the trends were generally dependent on the choice of regularization constant. Another unknown was the number and the type of the classifiers. We decided to use decision stumps where we evenly distributed equal number of them across the range of every dimension of data. As expected, the number of classifiers per feature also impacted the results. Unfortunately, increasing the number slowed down our algorithm significantly.

Even though most of our questions could have probably been answered through extensive testing for different parameter values, this was practically infeasible due to the bad running time of the algorithm. Because the algorithm is relatively new, we were unable to find publicly available implementation we could apply to our data or utilize when debugging. Professor Rudin was kind enough to send us some code snippets, but those were not sufficient to answer all of our questions. We tried filling in all the missing pieces, but we were unsuccessful because even with those snippets we could not reproduce the trends from the paper. The implementation we got from the author was significantly slower than ours so we were again faced with the unbearably long running times. We still wanted to see what would the effect of this algorithm be on our data, so we chose our best guess of the implementation and ran it. The results are presented in section 5.

5 Experiments

We developed a comprehensive test suite in order to evaluate the quality of regression and ordinal regression. The dataset we used has RGBD scenes with 3 and 4 objects. Each of the objects within the scene was from our object dataset. Since the computational complexity of ordinal regression is high, we had to carefully sample training points. We decided we will only focus on 15 (object, scene) pairs where the detector has a high chance of sampling a good pose (the performance depends on occlusion, noise, object type, etc.). For those pairs, we sampled 10 good and 10 bad poses per pair which yielded a total of 300 training points. Since validation is much faster, we could allow collecting more points. For each of the 15 (object, scene) pairs mentioned above, we obtained 1000 hypotheses, regardless of whether they are good or bad. This simulates the first round of correspondences in the object detection algorithm. As mentioned in section 1, the algorithm then ranks the hypotheses and picks only top 100 for refinement. Therefore, we decided to measure the quality of our ranking as a difference between the number of good kept hypotheses in old ranking vs. our ranking.

5.1 Results

5.2 Model Selection

We found w^* by finding the average number of good hypothesis classified after training w on one of our three training sets and testing on our three validation sets. For Regular Regression, we varied the regularization constant λ and the σ of the normal curve we are fitting the scores to. For Ordinal Regression, we varied the slack variable C. The following tables show the different parameter values and the number of hypothesis scored correctly versus using the basis $w = [1, 1, 1, 1]$ ones vector.

Model Selection Results

Regular Regression

Training Data 1:

λ	σ	V_1	V_2	V_3	**Avg**
0.0	0.1	-7.0	-5.0	-6.0	-6.0
0.0	**0.5**	**13.0**	**15.0**	**1.0**	**9.7**
0.0	**1.0**	**13.0**	**15.0**	**1.0**	**9.7**
0.1	0.1	-18.0	-14.0	-9.0	-13.7
0.1	0.5	-18.0	-14.0	-9.0	-13.7
0.1	1.0	-18.0	-14.0	-9.0	-13.7
0.5	0.1	-18.0	-14.0	-9.0	-13.7
0.5	0.5	-18.0	-14.0	-9.0	-13.7
0.5	1.0	-18.0	-14.0	-9.0	-13.7

Training Data 2:

λ	σ	V_1	V_2	V_3	**Avg**
0.0	0.1	0.0	-1.0	-3.0	-1.3
0.0	0.5	3.0	1.0	2.0	2.0
0.0	**1.0**	**15.0**	**18.0**	**9.0**	**14.0**
0.1	0.1	-22.0	-17.0	-17.0	-18.7
0.1	0.5	-22.0	-17.0	-17.0	-18.7
0.1	1.0	-22.0	-17.0	-17.0	-18.7
0.5	0.1	-22.0	-17.0	-17.0	-18.7
0.5	0.5	-22.0	-17.0	-17.0	-18.7
0.5	1.0	-22.0	-17.0	-17.0	-18.7

Training Data 3:

λ	σ	V_1	V_2	V_3	**Avg**
0.0	0.1	-5.0	-3.0	-4.0	-4.0
0.0	0.5	14.0	13.0	6.0	11.0
0.0	**1.0**	**17.0**	**16.0**	**9.0**	**14.0**
0.1	0.1	-22.0	-16.0	-16.0	-18.0
0.1	0.5	-22.0	-16.0	-16.0	-18.0
0.1	1.0	-22.0	-16.0	-16.0	-18.0
0.5	0.1	-22.0	-16.0	-16.0	-18.0
0.5	0.5	-22.0	-16.0	-16.0	-18.0
0.5	1.0	-22.0	-16.0	-16.0	-18.0

Ordinal Regression

Training Data 1:

C	V_1	V_2	V_3	**Avg**
0.01	-32.00	-28.00	-28.00	-29.33
0.10	-25.00	-21.00	-37.00	-27.67
1.00	7.00	14.00	13.00	11.33
5.00	14.00	28.00	19.00	20.33
10.0	**16.0**	**29.0**	**22.0**	**22.3**
15.00	12.00	23.00	14.00	16.33
20.00	15.00	29.00	18.00	20.67
25.00	14.00	27.00	15.00	18.67
30.00	15.00	24.00	19.00	19.33

Training Data 2:

C	V_1	V_2	V_3	**Avg**
0.01	-17.00	-13.00	-19.00	-16.33
0.10	1.00	8.00	7.00	5.33
1.00	30.00	47.00	43.00	40.00
5.00	40.00	59.00	45.00	48.00
10.0	**40.0**	**57.0**	**48.0**	**48.3**
15.00	40.00	58.00	44.00	47.33
20.00	40.00	58.00	38.00	45.33
25.00	39.00	54.00	34.00	42.33
30.00	43.00	55.00	34.00	44.00

Training Data 3:

C	V_1	V_2	V_3	**Avg**
0.01	-22.00	-20.00	-22.00	-21.33
0.10	-16.00	-16.00	-31.00	-21.00
1.00	18.00	27.00	5.00	16.67
5.00	21.00	30.00	1.00	17.33
10.00	22.00	39.00	10.00	23.67
15.0	**31.0**	**46.0**	**19.0**	**32.0**
20.00	17.00	25.00	-6.00	12.00
25.00	28.00	43.00	20.00	30.33
30.00	29.00	43.00	19.00	30.33

P-Norm Push

Training Data 1:

#Class	P	V_1	V_2	V_3	Avg
5	1	-58	-51	-60	-56.33
5	2	-14	8	10	1.33
5	8	-2	6	5	3
20	20	-123	-108	-102	-111
20	20	-70	-51	-36	-52.33
20	20	-32	-26	-13	-23.67

Training Data 2:

#Class	P	V_1	V_2	V_3	Avg
5	1	-52	-21	-21	-31.33
5	2	-71	-48	-52	-57
5	8	-78	-34	-48	-53.33
20	1	-58	-29	-40	-42.33
20	2	-66	-23	-35	-41.33
20	8	-39	-20	-25	-28

Training Data 3:

#Class	P	V_1	V_2	V_3	Avg
5	1	-21	-6	-16	-14.33
5	2	-19	11	-3	-3.67
5	8	-2	11	-9	0
20	1	-21	-2	3	-6.67
20	2	-24	-18	-9	-17
20	8	-35	-12	-10	-19

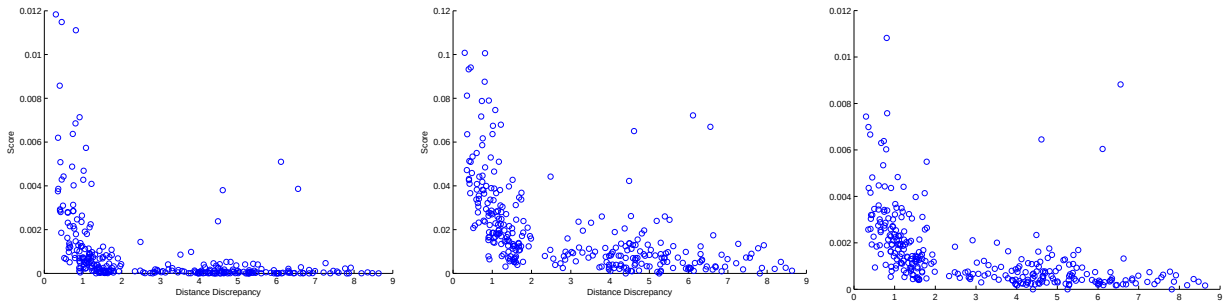
For completeness, this table contains the results obtained with P-norm push algorithm. However, as discussed in section 4, we could not verify the correctness of our implementation, so we will not base our conclusions on these results. The first column contains the number of classifiers used per one dimension of data. Column **P** represents the P-norm. Columns V_1, V_2 and V_3 show the relative change of performance versus the old scoring function.

5.3 Large Object Specific Results for w^*

From our model selection we found $w^* = [0.24, 0.12, 0.71, 0.01]$ from training on data set 2 with $C = 10.0$. The following table show the results of the weights w^* versus $w = [1, 1, 1, 31]$ per object. Each object has 1000 hypothesis generated; each set is ranked by $f(\phi) = \omega\phi$ and only the top 10% is stored. From this top 10% we note the number of good objects classified correctly. The 'ones' column shows the number of objects classified correctly in the top 10%; similarly, the 'ordinal' column shows the number of objects classified correctly in the top 10%. The third column shows the relative percentage increase per object.

Ones	**Ordinal**	**Relative Increase**
66.0	76.0	15.2 %
4.0	6.0	50.0 %
7.0	9.0	28.6 %
31.0	28.0	-9.7 %
10.0	7.0	-30.0 %
47.0	55.0	17.0 %
15.0	16.0	6.7 %
8.0	12.0	50.0 %
41.0	44.0	7.3 %
19.0	23.0	21.1 %
26.0	28.0	7.7 %
9.0	11.0	22.2 %
29.0	37.0	27.6 %
17.0	28.0	64.7 %
56.0	62.0	10.7 %
385	**442**	**14.805** %

Figure 2 something shows the performance of scoring versus the distance discrepancy. The following plots contain 150 good and 150 bad data points. The ω's used are the best found from the model prediction.



(a) $w = [1, 1, 1, 1, 1]$ (b) w from regular regression (c) w from ordinal regression

Figure 2: $\omega\phi$ vs. scalar discrepancy

6 Conclusion

The goal of our project was to find weights ω such that the scoring function $S = \omega\phi$ ranked hypothesis with smaller distance displacement D to have a higher score S . We implemented three approaches to train ω on our test data and verified the results with separate data files. We used separate training and validation data sets to make sure that our results were not over fitting. Furthermore, we created three training data sets to train ω with various parameters as described in our model parameter section.

6.1 Regular Regression

The tables in the model parameter section were used to find an optimal w^* for our scoring functions. For Regular Regression, the best results surprisingly were generated with no regularization factor ($\lambda = 0$). For the second and third training data sets $\sigma = 1$ yielded the best results, and for the first training data set $\sigma = 0.5$ and $\sigma = 1.0$ yielded the best results. Using a smaller value of σ or adding a regularization constant $\lambda > 0$ yielded results that were worse than just using $w = [1, 1, 1, 1]$ vector. We expected the regular regression method to be a baseline for this project and give only a slight improvement; the only unusual result was the lack of regularization factor yielding better results. Figure X shows the results of using w^* from training on data set 2 with $\lambda = 0$, and $\sigma = 1.0$. Compared to the ones vector (ref) the regular regression plot tends to elevate the low scores with $D < 2$ which correlated to classifying points slightly better than the ones vector.

6.2 Ordinal Regression

Ordinal Regression provided the best results of the three approaches we tried. In our model selection to find the best ω^*, we varied the slack variable C from [0.01,30]. For the first two training data sets $C = 10$ yielded the best results, whereas the third data set did best with $C = 15$. Ordinal Regression yielded the best results from our model selection with Training Data 2 and $C = 10$, which generated $w^* = [0.24, 0.12, 0.71, 0.01]$. Table X uses this weight vector ω^* to show the relative percent increase for each object in a test data set. Overall $/w^*$ showed a 14.805% increase to the ones vector; however, the performance increase varied for the specific object. The range of the relative percent increase was very surprising: the largest decrease was -30% increase, whereas the maximum increase was 64.7% increase. Figure 2(b)) shows the results of using w^*. Similar to plot b the low scores with $D < 4$ are elevated which clearly correlated to the better performance.

6.3 P-Norm Push

Even though we could not confirm the correctness of our P-norm push implementation, we decided to try it against our dataset. Despite the fact that we could not reproduce the trends from Rudin's paper, the predicted trends were mostly clear in our dataset. Small **P**s had significantly worse performance than the vector of ones. We noticed that our results are much better with the smaller number of classifiers per feature. This might be due to the fact that we limited the number of boosting rounds to 20. This number is probably low, but due to the long running time, we had to sacrifice performance for practicality. The only moot point about our implementation is the way we dealt with infinities mentioned in section 4.2. We set a manual upper bound on $|\alpha|$ to be 25 which is significantly higher than the values of α we obtained in the case without infinities. We have tried other ways of dealing with infinities, but they all had worse performance. Somewhat

unsurprisingly, that was the main point of confusion when implementing the algorithm.
In our experiments, P-norm push did not perform nearly as well as we expected, but we believe that we can obtain improve the results with better implementation.

6.4 Summary

One of the challenges with this project was dealing with actual data that was extremely noisy with lots of outliers. We predict that less outliers would have made regular regression a much stronger candidate. Since we had such noisy data, we needed to use lots of data to get the best results; however, this wasn't a feasible solution because both the ordinal regression and P-norm push scale the training data quadratically. By using SVM^{rank} we were able to use more data for training than with our Ordinal Regression Matlab implementation with relative successful results. We were very concerned about over-fitting to the scenes and object models we are using, which is why we used separate training sets and large validation sets.

We noticed that our results differ significantly from one training dataset to another. We believe this is mostly due to the fact that our training sample is too small so we depend on the luck of the draw. It is interesting that different methods prefer different training datasets which speaks for their intrinsic differences.

6.5 Future Work: Data Generator

As we can see from the table in section 5.3 (Large Object Specific Results Section) the improvement from the learned weights varied significantly across the objects. This lead us into thinking that learning specialized weights for individual objects might lead to even better results. However, this would be hard to train because it would require a great amount of data. Obtaining point-clouds with labeled true poses is very tedious and time consuming. Our plan for the future work is to develop a generator for synthetic point clouds. We have filtered point clouds of individual objects so we can easily take a random sample of those and artificially merge then into a scene. We only need to be careful not to break any physical constraints (e.g. we cannot have overlapping objects or objects that are not in stable balance). Additionally, we need to make sure to select a viewpoint and remove all the points that would have not been visible from there. While this data would not necessarily be natural (e.g. we rarely find an alarm clock next to a banana), it would still provide us with a decent baseline for training.

6.6 Division of Labor

Regular regression: Ariel and Sanja

Ordinal Regression: Ariel

P-norm Push: Sanja

Results, Conclusion, Write-Up: Ariel and Sanja[**?**]

7 Acknowledgements

The authors would like to acknowledge

- Jared Glover for providing assistance and advice on using the object detector (along with labeling the data sets).
- Cynthia Rudin, PhD for responding to our inquires about her paper.
- Lawson Wong and Owen Macindoe for Matlab and LaTeX advice.
- Learning and Intelligent Systems Research Group