Objective Detection and Nearest Neighbor Search-LSH

Jiahao Yang
Department of Statistics
University of Washington
Seattle, WA 98195
yangjh39@uw.edu

Abstract

In this project, I work with the MS-COCO dataset to complete the Object detection/localization and Nearest Neighbor search/retrieval tasks. For object detection, I use linear-SVM classifier with squared hinge loss function to classify each category and use hard negative mining method to deal with the unbalanced data. The mean AP score on test set is about 0.3483. For Nearest Neighbor search, I build Locality Sensitive Hashing(LSH) data structure with Euclidean distance and query K nearest neighbors based on this data structure. The mean AP score on test set is about 0.3802.

Keywords: MS-COCO, Objective detection, linear-SVM, LSH, Mean AP score

1 Introduction

The MS-COCO dataset is one of the current state-of-the-art datasets used for developing and benchmarking in several computer vision including (a) semantic segmentation, (b) object localization and detection, (c) multi-label learning, (d) image retrieval, amongst other interesting and practically relevant questions. The data I will use includes 2 super-categories of MS-COCO dataset: vehicle and animal and there are 18 categories under these two super-categories in total.

For object detection task, I need to build an object detector which will comprise of a binary classifier for each category. Given features of an image patch corresponding to a bounding box, the classifier need to know whether this patch contain an object of the category of interest.

For Nearest Neighbor(NN) search, I will create LSH data structure for approximate NN. And for a given validation or test image patch, I need to first find the bucket that it belongs to and then find the K nearest image patches from the set of training image patches which are also in this bucket.

2 Methodology

2.1 Data preprocessing

2.1.1 Region proposal

The first step is to identify patches in an image that are likely to contain objects. Here I follow the instruction to use a region proposal algorithm known as selective search(SS). Selective search essentially returns a large number (say, 1000 or 2000) of bounding boxes corresponding to all patches in an image that are most likely to be objects. These region proposals can be noisy, overlapping and may not contain the object perfectly but amongst all these region proposals, we usually have one proposal which will be very close to the actual object in the image.

2.1.2 Extracting features of an image patch

When I have a bounding box, I need to extract features of the image patch corresponding to a bounding box. Here I use the code provided to extracts almost 10K dimensional feature vector per image patch. I use adaptive max pooling to achieve this.

2.1.3 Extracting ground truth labels

The next step is to extract ground truth labels for the locations of these image patches I get from SS. Here I use COCO API to access the ground truth bounding boxes in each image. Then I calculate the intersection-over-union(IoU) between these truth bounding boxes and the image patches and set the image patches' labels to be truth if its IoU is larger than 0.5.

The IoU between two rectangles r1 and r2 is,

$$IoU(r_1, r_2) = \frac{Area(r_1 \cap r_2)}{Area(r_1 \cup r_2)}.$$

2.1.4 Implement

Since there are a huge number of bounding boxes in total, it is impossible to store all the features in memory. Thus, in data preprocessing, my goal is to generate a table that contains the truth ground labels, the image id and the bounding box id of each bounding box. Since there are 18 categories, the truth ground label is a vector of length 18 which only contains 0 and 1 and 1 means this bounding box is belong to the respective category. Thus this table will have n rows and 20 columns, where n is the total number of bounding boxes.

Algorithm 1 Data preprocessing

Input: Location of bounding boxes in each image;
Output: The table contains truth ground labels, the image ids and the bounding box ids;
for (b, i) $\in$ (candidate bounding boxes, respective image id) **do**
  Calculate IoU between b and true bounding boxes in image i;
  if $IoU > 0.5$ **then**
    c = the category id of the true bounding boxes;
    Set the c_{th} vector element to be 1;
  end if
  Set the 19_{th} and 20_{th} vector elements to be i and b;
  Store this vector in table.
end for

After data preprocessing, we have build a table contains labels, image id and bounding-box id for each bounding box. Based on this table, instead of storing features of all bounding boxes, we can only extract features for the bounding boxes that we will use to train and evaluate my model.

2.2 Object detection

2.2.1 Classifier - linear SVM

SVM is a supervised machine learning algorithm which can be used for classification or regression problems. Although in this experiment, I choose to use linear SVM, SVM can uses a technique called the kernel trick to transform the data and then based on these transformations it finds an optimal boundary between the possible outputs. This model is flexible to deal with complex problem and it is convenient for me to modify my model to a non-linear model.

I use a little trick to improve my loss function of linear-SVM model. Instead of using hinge loss function, I use squared-hinge loss function which is

$$l(x, y) = \frac{\lambda}{2} \parallel w \parallel + \frac{1}{2n} \sum_{i=1}^{n} (max(0, 1 - y_i(w \cdot x_i)))^2.$$

The reason why I do this is that the although the hinge loss is convex, it is not smooth. To improve the performance of using gradient-descent-based method, such as Heavy Ball, I square the hinge loss such that the loss function becomes convex and smooth.

To train my classifier, I use Heavy Ball method. And in order to get the best parameters of my model, I use grid search to choose my parameters based on the train AP scores on validation set. Finally, my parameters are set to be regularization parameter $\lambda = 5$, learning rate $\gamma = 5 \times 10^{-6}$ and mini-batch size $b = 16$.

2.2.2 Subsampling negatives

As I noticed before, the classification problem is pretty unbalanced. There are a small number of positives (corresponding to the rectangles with $IoU > 0.5$ with some ground truth bounding box) and a large number of negatives (all others). Thus, I sub-samples negatives to have 5 negatives per positive to ensure that my models learns something meaningful.

2.2.3 Hard negative mining

When I retrain my model using hard negative mining, I first choose the first n negative examples from the negative examples which are assigned to be positive with highest probability in previous training set and then randomly sample 4n negative examples to construct my 5n negative examples.

This method is reasonable because when we choose the negative examples which are assigned to be positive with highest probability, I actually choose those points which our model does not learn from them a lot. And I retrain our model on these points so that more information in these points can be learned by our model. And we choose n new negative examples so that our model can learn the negative points' information from new points in case overfitting.

2.2.4 Implement

The whole algorithm is shown below. Since I need to get 18 classifiers, I use the Python package "multiprocessing" to parallel my program. In order to get the best parameters, I use grid search for one classifier and then I use the same parameters for other classifiers.

Algorithm 2 Object detection with hard negative mining - train

Input: The table contains truth ground labels, the image ids and the bounding box ids;
Output: 18 classifiers for 18 respective categories;
for $c \in$ 18 categories **do**
  n = the number of positives of category c;
  Subsampled $5n$ negatives' image id and bounding boxes id from the table we already have;
  $data$ = features of these n positives and $2n$ negatives extracted based on ids;
  for $i = 1, 2, \cdots, iteration\ \ number$ **do**
    Train linear-SVM classifier using $data$;
    d_1 = n positives in $data$;
    d_2 = n negatives in $data$ which are assigned to be positive with highest probability;
    id_3 = $3n$ negatives' ids which are randomly sampled from the table we already have;
    d_3 = features of new sampled $3n$ negatives extracted based on id_3;
    Update $data$ to be the combination of d_1, d_2 and d_3
  end for
end for

2.3 Nearest neighbor search

2.3.1 Locality Sensitive Hashing(LSH)

In order to realize nearest neighbor search, at first, I need to create my data structure of choice for approximate NN search over patch features corresponding to given bounding boxes from the training set.

Here I use LSH. For each bounding box, I choose L functions g_j where $j = 1, 2, \cdots, L$ by setting $g_j = (h_{1,j}, h_{2,j}, \cdots, h_{k,j})$ where $h_{i,j}$ is hash function. I define $h_{i,j}$ like below.

$$h_{i,j} = \lfloor \frac{a \cdot x + b}{r} \rfloor,$$

where $a \in \mathbb{R}^d$is randomly picked from the Gaussian distribution and $b \in U\,[0, r)$.

Since it is impossible to load features of all the bounding boxes, I embed the process of extracting features into the part of building LSH. I only load one image's features and then construct a part of hash table based on these features and then extract next image's features to construct the next part of hash table.

2.3.2 Retrieve

To find the K nearest neighbors for a given val or test image patch, what I need to do is to find the K nearest image patches from the set of training image patches using LSH data structure.

Suppose I need to find the K nearest neighbors for a test bounding box q, at first I will get L key vectors $g_j(q),\ \ j = 1, 2, \cdots, L$ whose length is k. Then for each $g_j(q)$, retrieve the points from the bucket $g_j(q)$. I subsample 20% points in this bucket and calculate the distances between q and these points. And then, I choose K points with smallest distances. Finally I calculate the score for a class C which is (number of retrieved patches that belong to class C) / K to get the AP score of class C. To avoid having multiple overlapping patches, I only retrieve one patch per image.

To deal with the big data problem, I extract features of bounding boxes only when I need to calculate the distance between them.

2.3.3 Implement

The process of building LSH and retrieving neighbors for one test bounding box is shown below.

In order to find the best parameters, at first I choose initial parameters based on posters' experience. And then I just change one parameters at one time and to see whether I need to increase this parameters or the decrease this one. And then I will keep the parameter I have tuned and to find the other best parameters in the same way.

Algorithm 3 Nearest neighbor search - LSH

Input: Number of buckets L, Length of hash vector k, Scale value r;
Output: Local Sensitive Hashing Table;
p = length of features for a bounding box
Generate L $k \times p$ random matrices;
for $i \in$ image **do**
    Extract features for all the bounding boxes of image i;
    for $j \in$ bounding box in image i **do**
        Get hash table g(j) based on the features and L random matrices we already have;
    end for
end for
Store LSH table and L random matrices

Algorithm 4 Nearest neighbor search - retrieve

Input: LSH table h, number of neighbors K, test bounding box q;
Output: K nearest neighbors of q;
f = features of bounding box extracted;
calculate $g_j(q)$ = bucket keys of q based on L random matrices we have, where $j = 1, 2, \cdots, L$;
D, use this variable to store distances between q and candidate bounding boxes;
for $j = 1, 2, \cdots, L$ **do**
  if $g_j(q) \in h$ **then**
    d = data points in bucket $g_j(q)$;
    d_{sub} = data points subsampled from d;
    $dist$ = distances between q and d_{sub};
    Store $dist$ in D
  end if
end for
Find the K points with smallest distances from D

3 Result

3.1 Object detection

3.1.1 Optimize plots

The results of losses and AP scores on train set and validation set when I train and improve my classifier using hard negative mining are shown below.

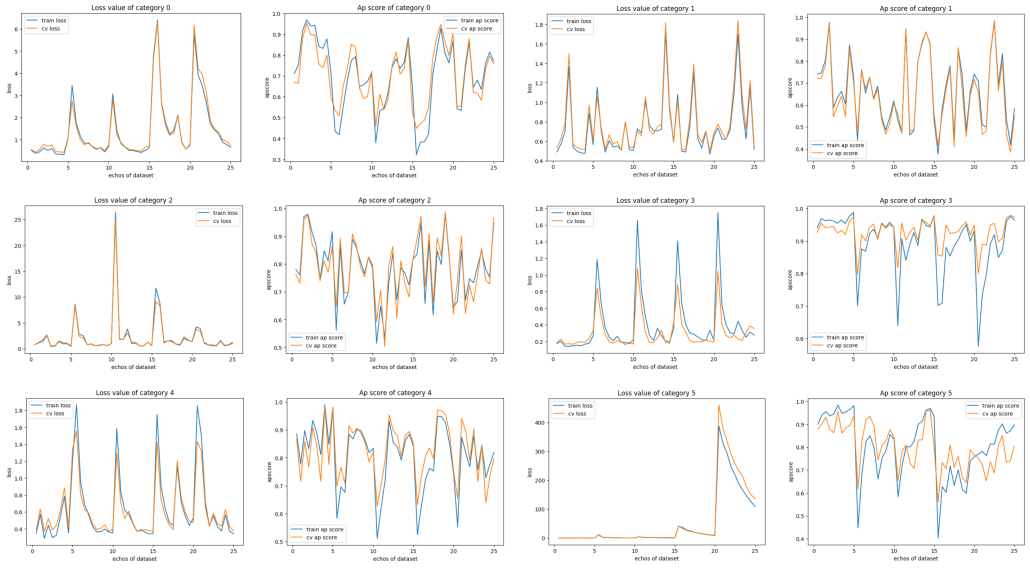

Figure 1: Loss and AP score of category 0-5

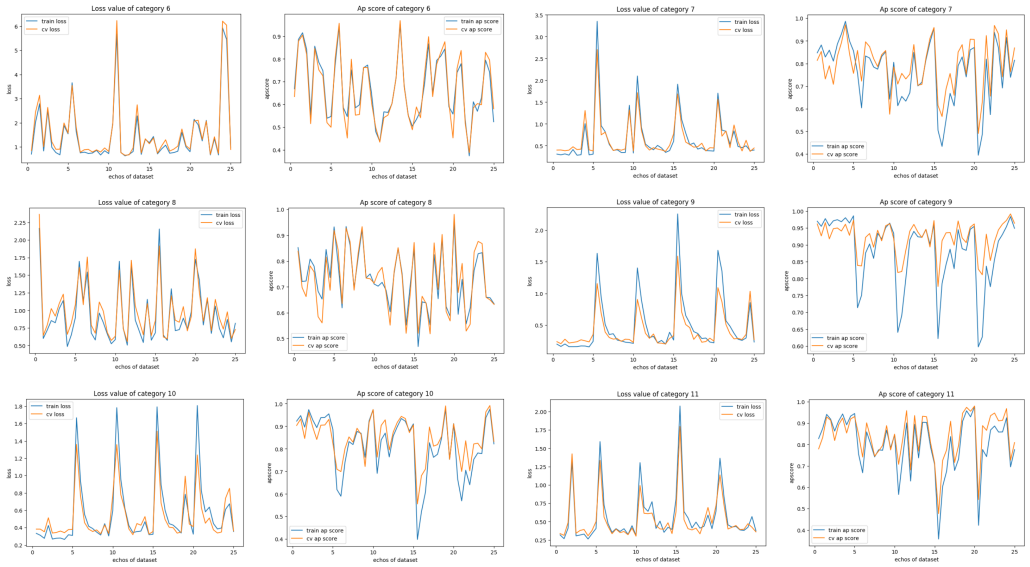

Figure 2: Loss and AP score of category 6-11

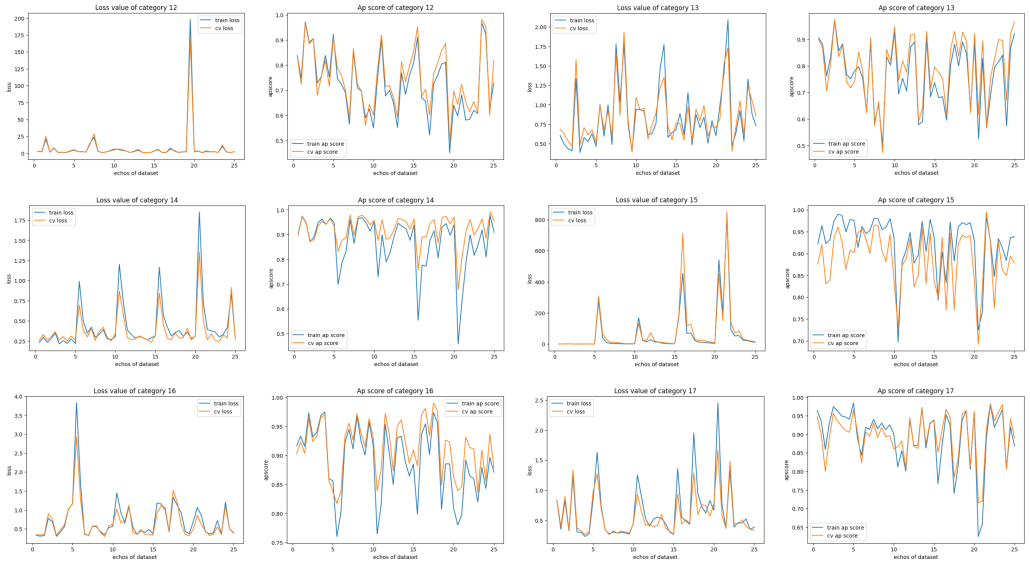

Figure 3: Loss and AP score of category 12-17

3.1.2 Performance matrix

The AP score for each categories on test set is shown in the table below.

Table 1: AP scores of objective detection on test set

Category	0	1	2	3	4	5
Mean AP score	0.087236	0.280792	0.306912	0.484109	0.467137	0.156474
Category	6	7	8	9	10	11
Mean AP score	0.058436	0.14062	0.186615	0.292493	0.928786	0.262626
Category	12	13	14	15	16	17
Mean AP score	0.110148	0.156377	0.071819	0.742532	0.598441	0.940618

And the mean AP score on test set is 0.3483.

3.2 Nearest neighbor search

3.2.1 Performance plots

To plot mAP vs K (for a fixed setting of hyper-parameters) for different values of K from {10, 50}, I choose $L = 10, k = 40, R = 200, c = 2$. The result is shown below

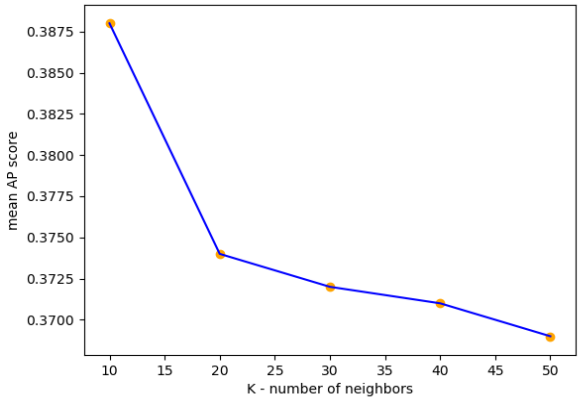

Figure 4: Mean AP score vs. number of neighbors on testset

To plot average distance to the K nearest neighbors vs search time and mAP vs the search time, by varying the hyper-parameters of your approximate NN algorithm but keeping K fixed, I choose $K = 10$. The results are shown below.

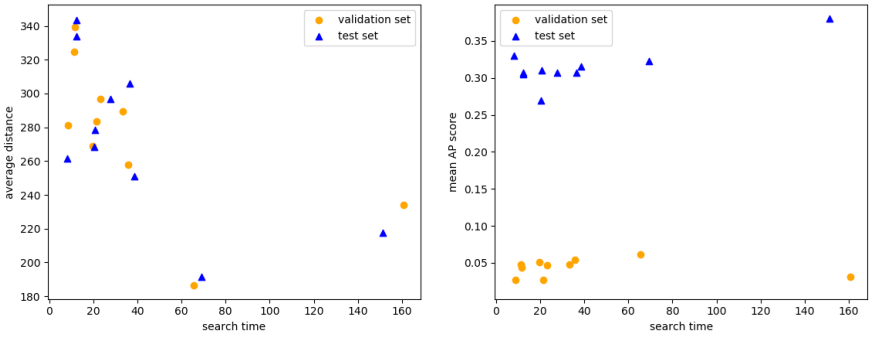

Figure 5: Average distance/mean AP score vs. search time on test and validation set

3.2.2 Performance matrix

The AP score for each categories on test set is shown in the table below.

Table 2: AP scores of nearest neighbor search(LSH) on test set

Category	0	1	2	3	4	5
Mean AP score	0.210725	0.255852	0.761848	0.337298	0.627363	0.107021
Category	6	7	8	9	10	11
Mean AP score	0.324751	0.217325	0.217507	0.262319	0.234231	0.585849
Category	12	13	14	15	16	17
Mean AP score	0.250054	0.559442	0.445626	0.464818	0.486518	0.459245

The mean AP score on test set is 0.3802.

4 Conclusion

4.1 Object detection

The loss value and AP score on train set and validation set of all categories with hard negative mining are shown in Figure 1, 2 and 3. AP scores are shown in Table 1. We can find that when we use hard negative mining, the mean AP score on validation set will decrease first, then it will increase. This is reasonable since at first the data that the model does not learn very well will hurt the model's performance. With more data trained, the model is improved by reducing over-fitting and thus performs better on test set.

4.2 Nearest neighbor search

Based on Figure 4, we can find that with K increasing, the mean AP scores on both data set decrease. This is because when K becomes larger, we will explore more possible bounding boxes. Since in my algorithm, I choose the best K neighbors to calculate the AP scores, we have more opportunities to find the neighbors which are not the best points. Thus these "worst" neighbors will hurt our mean AP score.

Based on Figure 5, I notice that although the 4 parameters(L, k, c, R) may be different for each point, there is still a trend that with the increase of search time, the average distance to K nearest neighbors will decrease. This is true since more search time means we explores more possible neighbors which means we have larger possibilities to find the best K neighbors. This will lead to the decrease of average distance.

For the mean AP score vs. search time plot, although it is not very obvious, on the test set we still can find that with the increase of search time, the mean AP score will increase slightly too. The possible reason is the same as the decrease of mean distance.

References

[1] Fei-Fei Li, Andrej Karpathy & Justin Johnson (2016) cs231n, Lecture 8-Slide 8, Spatial Localization and Detection.

[2] T. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollar, L. Zitnick. "Microsoft COCO: Common Objects in Context". ECCV. 2014.

[3] Girshick, Ross, et al. "Rich feature hierarchies for accurate object detection and semantic segmentation." Proceedings of the IEEE conference on computer vision and pattern recognition. 2014.