Object Detection and Retrieval by Linear Regression Classifier and Locality-Sensitive Hashing (LSH)

Lun Jiang
Department of Applied Mathematics
University of Washington
Seattle, WA 98195
lunj@uw.edu

Abstract

This final project is focused on object detection and retrieval on MS-COCO dataset. A multi-category linear regression classifier and a nearest neighbor data structure based on locality-sensitive hashing were implemented for detection and retrieval, respectively. The results of object detection and retrieval, along with implementation details like hard negative mining and e2LSH will be discussed.

1 MS-COCO dataset

The data used in this project is a part of MS-COCO dataset. COCO, aka. Common Object in Context, is a large-scale dataset with images belonging to different categories [1], which is wildly used in object detection and retrieval researches and projects. This project is focused on 18 categories of objects in COCO dataset: airplane, bear, bicycle, bird, boat, bus, car, cat, cow, dog, elephant, giraffe, horse, motorcycle, sheep, train, truck, zebra.

Two types of dataset are used as sources: tiny dataset and small dataset, both of which are sampled from COCO dataset, different in total number of images and distribution of categories.

This project uses localized bounding boxes of objects to extract data from images in dataset. The image patch bounded by bounding boxes are then featurized into feature vectors to be taken as input for object detection or object retrieval.

Two kinds of features are used to generate data vectors: tiny and small, both of which are the numerical data featurized from images, different in the length of featurized data vectors, i.e. 1472 for tiny and 11776 for small.

2 Object detection

The object detection work of this project is done by a simple but effective linear regression classifier, of which the layout, tuned parameters, and results will be discussed in this section.

2.1 Linear regression model

This structure of the linear regression model is shown in Figure 1. The model can be considered as a singe layer neural network with regularization. Here the objective function to optimize is:

$$L(w) = \frac{\lambda}{2} \|w\|^2 + \frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{k} l(y_{ij}, f_{ij}(w)) \tag{1}$$

Submitted to University of Washington course CSE 547 Machine Leaning for Big Data. Do not distribute.

The loss function $l(y, \hat{y})$ is logistic:

$$l(y, \hat{y}) = y \log(1 + \exp(-\hat{y})) + (1 - y) \log(1 + \exp(\hat{y})) \tag{2}$$

Stochastic gradient descent (SGD) with proper mini-batch size is employed when training the model. The detailed set-up of the model and the results will be presented later in this section.

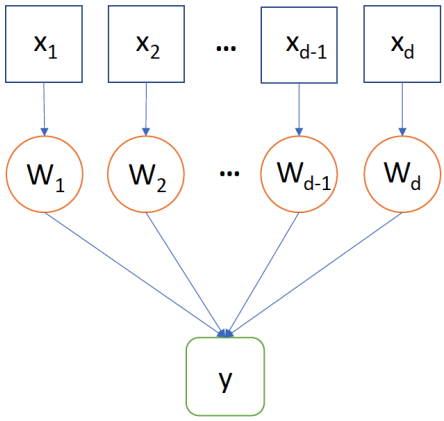

Figure 1: Structure of the linear regression model.

2.2 Sampling strategies

Appropriate sampling and sub-sampling of the dataset following certain strategies are necessary in this project, due to several concerns discussed below.

First, the total size of the input data for this classifier is enormous, and the input data is composed of a relatively small number of positive boxes with an extremely large number of negative boxes. Sub-sampling of negatives is a proper strategy to save computing time while maintaining efficiency. A manually chosen ratio between the number of positive samples and negatives (usually between 1:2 to 1:8) is applied during sampling to set constrains to the number of negative samples.

Meanwhile, for the positive boxes, there is high probability for the generated bounding boxes in each image to overlap each other. After featurization (projection into data vectors), the overlapped boxes may produce the same data vector, which leads to duplicated data and resultant redundancy in the input dataset. So we should also remove those highly overlapped boxes during sampling. To give "highly overlapped boxes" a clear definition, a threshold IoU of 0.5 is set as the margin of two distincted boxes, i.e., two boxes with IoU greater than 0.5 would be considered "similar boxes" and one of them will be removed to avoid redundancy.

One more thing to notice, the classifier is binary, and the classification work are done per category, which means the positives and negatives for the 18 different categories should be different. For example, when classifying the category "giraffe", only those boxes with IoU greater than 0.5 with a ground true "giraffe" box will be considered as a positive sample, and a box containing "dog" patch will be considered as a negative box, just like a box containing background. This means when the current working category is changed, re-sampling of positives and negatives is necessary.

The complete sampling process is listed below:

(1) Choose one category from the 18 categories of interest.

(2) Loop through all image in the dataset, localize the ground true boxes.

(3) Within the same loop as above, figure out all positive boxes by computing IoU to the ground true boxes to get all positive candidates.

(4) Among all positive candidates, compute IoU pariwisely and remove highly overlapped boxes. The results of this step is used as the positive box samples.

(5) Within the same loop above, figure out all negative boxes and add them to an enormous negative pool.

(6) Randomly sample from the negative pool and add to negative box samples, until the positive-to-negative ratio reaches the manually set ratio.

(7) Project the positive and negative samples to data vectors and perform binary classification.

(8) After finishing one category, move on to the next and redo all steps above until finishing the last category.

The process above is applied to construct the train and validation data. For the test set, neither sub-sampling of negatives nor removing redundancy will be performed, which means every single positive and negative boxed image patch will be used to generate input data vectors. The goal is to keep the data distribution in the test set unchanged regardless of the sampling strategies, providing a unique criterion to test the implementation of the classifier.

2.3 Memory issues

When dealing with big data, memory issue is one important concern. For example, if small feature is used to construct data vectors, an estimation of input data size can be obtained as following: The size of a double-precision floating point number is 16 bytes, and a data vector with small features contains 11776 doubles. Use the test set as example, which contains in total about 770,000 projected data vectors from bounding boxes without sub-sampling of negatives or removing redundancy. Then the estimated total input data size of the test set can be computed by: $16 \times 11776 \times 770000 = 135$ GB. Since 135 GB is a large amount of memory, it may cause significant memory issues when loading the entire test set data into memory.

To solve this problem, appropriate parallelism can be helpful. Loading the data by parts and discard old data parts may also be useful. In this project, to circumvent this memory issue, tiny features with small dataset are used as the input data to train the model and calculate the AP score per category as well as the mAP, the results of which will be shown later in this section.

2.4 Parameter tuning

Since the structure of linear regression model is relatively simple compared to multi-layer perceptron (MLP) with many layers. The number of hyperparameters and parameters is relatively small. So tuning this model is not so difficult compared to tuning MLP or even more complicated models. After several trials with parameter tuning, the following set of parameters is commonly used in this project to train the model and calculate test APs:

(1) learning rate $r = 1 \times 10^{-5}$

(2) minibatch size = 5

(3) regularization factor $\lambda = 10$

(4) total step limit $N = 10000$

To save computing time and avoid overfitting, the training process will be stopped earlier before total step limit, based on the loss and AP scores on the validation set. The training process stops when the val loss converges or when it reaches the lowest loss point before bouncing back. Usually this point is coincident with the maximum AP point of the validation set. Then the test AP will be calculated based on the weights on that point.

2.5 Hard negative mining

Hard negative mining is a technique to improve the performance of the classifier by re-train the model with hard negatives. This technique can be help in this project since the number ratio between positive boxes and negative boxes in the test set is very low, i.e., there are far more negative samples than positive samples, which may lead to a significant number of false positive detections in the classification results.

The basic idea of hard negative mining is to re-train the model again and again with "hard negative" samples which leads to false positive predictions, until they are correctly classified as negative samples.

In this project, hard negative mining is performed automatically in code during training process. There is a boolean value as a switch to control whether to do hard negative mining or not per certain epoch. If the switch is on, then the epoch value can be specified by another input argument.

The complete hard negative process is listed below:

(1) Construct the train set data with positive and negative samples according to the *positive-to–negative ratio* (e.g. 1 : 8).

(2) Train the model until the train set loss and AP score show convergence.

(3) Make prediction on train set data with current model weights.

(4) Collect all false positive predictions, and mark the input data samples leading to false positive predictions as "hard negative samples".

(5) Re-sample the train set data: first add all positive samples into the re-sampled train set.

(6) Then add all hard negative samples, which give false positive predictions, to the re-sampled train set.

(7) And then retrieve negative samples one by one randomly from the huge negative pool constructed earlier. This step is completed when the ratio between positive and negative samples in the re-sampled train set reaches the *positive-to–negative ratio* (e.g. 1 : 8).

(8) Use the re-sampled train set as new train set and continue training process.

(9) Redo the steps above until there is no more hard negative sample in the train set.

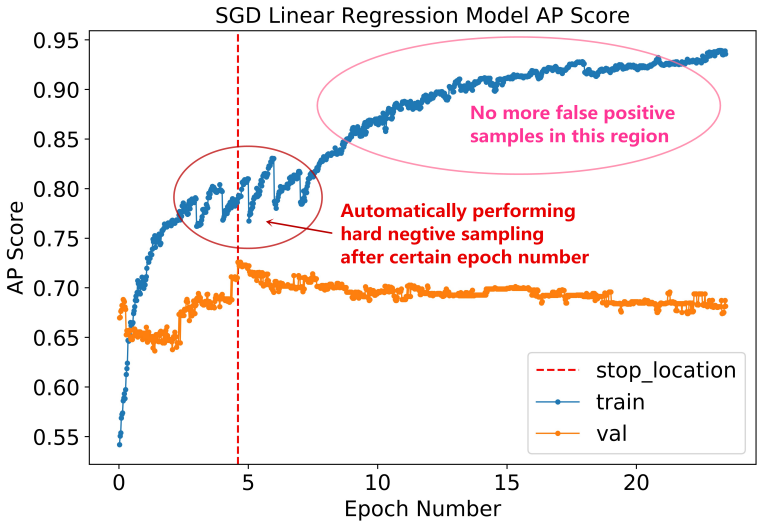

Figure 2: Example of hard negative mining.

An example of doing hard negative sampling during training is show in Figure 2, which shows the curve of train and val set AP scores during the training process. In this case, hard negative mining is performed per each epoch after convergence on train set. Oscillations with serrated shape were observed indicating the effect of hard negative mining: the AP score decreased sharply after re-sampling and then bounced back quickly. The hard negative sampling process was repeated until there were no more false positive predictions on train set.

As is shown in Figure 2, the train set AP score increased after a few rounds of hard negative mining and no more false positives were observed. However, it seemed like hard negative learning can only improve the model performance on train set, without obvious influence on the val set, since there is no significant performance improvement observed on the val set AP score curve.

Moreover, based on the model performance on the test set, it seems that sometime hard negative mining leads to improvement on test set AP, sometime not. The behavior is somehow random, so there is no persuasive conclusion drawn in this project about the influence of hard negative mining on improving prediction accuracy.

2.6 Results and discussions

Figure 3 shows the results of AP scores per category and mAP of all 18 categories. The results were obtained from small dataset using tiny feature. As is shown in the Figure 3, generally the mAP is around 38%.

Category	AP Score
airplane	0.100943865
bear	0.021800781
bicycle	0.679131258
bird	0.948678692
boat	0.978152119
bus	0.128916376
car	1
cat	0.092665129
cow	0.492040981
dog	0.326935452
elephant	0.130464173
giraffe	0.120860096
horse	0.233345526
motorcycle	0.131103281
sheep	0.544946483
train	0.025418378
truck	0.67561302
zebra	0.205969348
mAP	**0.379832498**

Figure 3: AP scores of object detection by linear regression classifier.

The AP score results for some categories may appear somehow "weird", for example, the AP scores for bird, boat, car are extremely high; while the AP scores for bear, train are extremely low. The appearance of such behavior may be ascribed to the way how AP score is calculated.

The new protocol computes the AP score only over the ground-truth positives and the detections made by the algorithm, which means only true positive and false positive predictions will be considered. Under this protocol, if a category has very few true positive sample, and the model also gives very few positive detections, then its AP might be extremely high if those detections were predicted correctly.

For those categories with extremely low AP score, the reason may be due to an enormous number of false positives predictions along with very few true positives. In this case, the model misclassified a large number of negatives as positives. Perhaps doing more hard negative mining can help to avoid such behavior, based on the discussions in the previous section.

3 Object retrieval

The object retrieval work of this project is done by a K-nearest neighbor data structure constructed by locality-sensitive hashing (LSH), of which the theoretical background, algorithm, parameters, and results will be discussed in this section.

3.1 Locality-Sensitive Hashing

Locality-sensitive hashing (LSH) is an algorithm to reduce the dimensionality of high-dimensional data using hash functions mapping “similar” data into same buckets.

Random projection is one common way to construct LSH functions due to its distance preservation property: if two data points are close in the original space, then they are highly possible to be still close in the projected subspace, with probability given by the *Norm Preservation Theorem*: for all

$x \in R^d$, the norm of the random projection $\phi(x)$ approximately maintains the norm of the original x with high probability:

$$P((1-\epsilon)\|x\|^2 \leq \|\phi(x)\|^2 \leq (1+\epsilon)\|x\|^2) \geq 1 - 2e^{-(\epsilon^2-\epsilon^3)m/4} \tag{3}$$

3.2 LSH functions

There are a few kinds of LSH functions used or tried in this project, here are two examples:

(1) Bitwise Sampling: Bitwise sampling LSH maps high dimensional data vectors to Hamming space. It is one kind of random projection using binary information obtained from each projection operation. A series of randomly generated hyperplanes out of Gaussian distribution are employed to project the data vector by the $sign(x)$ function. For each data vector $\mathbf{p}$, the binary projection given by a hyperplane $\mathbf{u}$ can be shown as:

$$h(\mathbf{p}) = sign(\mathbf{u} \cdot \mathbf{p}) \tag{4}$$

Each projection gives exactly one bit of information. If k hyperplanes are used to project one data vector, the k-bit results can form a binary number of k-bit length. And then the k-bit binary number can be used as the key to the original data vector in the LSH hashtable. The geometrical graph in Figure 4 gives an intuitive explanation about how this algorithm work to maintain locality in the projected Hamming space. Suppose there two data vectors $\mathbf{p}$ and $\mathbf{p}'$, which are adjacent in the high-dimensional space (use 2-d for simplicity in this graph). A hyperplane $\mathbf{u}$ is employed to project the two vectors. In a 2-d space, there are 4 possible regions I, II, III, IV that the hyperplane $\mathbf{u}$ might be located. If the hyperplane $\mathbf{u}$ is located at regions I and III, the $h(\mathbf{p})$ hash function would map the two data vectors to the same bit 0 or 1, maintaining the adjacency in the projected Hamming space. And the probability for the above case to happen is:

$$Pr(h(\mathbf{p}) = h(\mathbf{p}')) = 1 - \frac{\theta}{\pi} \tag{5}$$

where θ is the angle of the two data vectors in the original high-dimensional space. This means the closer of the vectors in the original space, the higher probability that they will be projected to the same digit, thus it satisfies the property of locality.

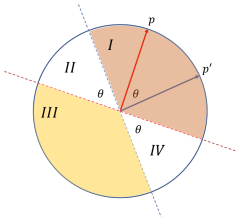

Figure 4: Geometrical explanation of bitwise sampling.

(2) e2LSH: Euclidean LSH, aka. e2LSH, is another kind of LSH function. The basic idea of e2LSH is similar to bitwise sampling LSH in maintaining locality. However, unlike bitwise sampling, it maps the data vector into a set of hash buckets rather than binary number in Hamming space [2].

$$h_{\mathbf{x},b}(\mathbf{p}) = \left\lfloor \frac{\mathbf{p} \cdot \mathbf{x} + b}{w} \right\rfloor \tag{6}$$

where $\mathbf{x}$ is a randomly generated vector to project the data vector, just like the hyperplane in the bitwise sampling case. A number of k random vectors are used to project the original data vector in serial, and the k outcomes

$$H(\mathbf{p}) = [h_1(\mathbf{p}), h_2(\mathbf{p}), ..., h_k(\mathbf{p})]$$

are then combined into one fingerprint $T(\mathbf{p})$ through bucket hashing, which is implemented by the inner product and modulo division below, where $\mathbf{W}$ is a random weight vector and P is a big prime number:

$$T(\mathbf{p}) = \mathbf{W} \cdot H(\mathbf{p}) \mod P \tag{7}$$

The fingerprint is then used as key to the original data vector in the LSH hashtable.

Both bitwise sampling and e2LSH functions were tried in this project, and the results presented in this report were obtained by e2LSH due to better accuracy. The denominator in e2LSH function is set to $w = 1$, and the big prime number is set to $P = 2^{32} - 5$ throughout the project.

3.3 K-nearest neighbor structure

The object retrieval work in this project is done by an LSH-based K-nearest neighbor data structure. This structure is essentially a number L of different LSH hashtables.The construction and querying processes of the data structure are shown below:

For the construction process:

(1) Construct L groups of LSH functions $g_1, g_2, ..., g_k$ each of length k. Each group of LSH functions produce a hashtable, so there will be a total of L hastables.

(2) Loop through all date vectors $\mathbf{p}$ in the data set. Use the L groups of LSH functions to project the date points into the corresponding hashtable.

For the querying process:

(1) Check all the hashtables in order, looking for cR nearest neighbors for the query point. Here cR is the cutoff distance to justify a neighbor.

(2) Among all cR nearest neighbors, find the k nearest neighbors and use their information to predict the category of the querying point and calculate AP score vector.

(3) If after 3L points have been checked, and the algorithm have not found a cR point, then the query point is believed to have no nearest neighbor in the data set, thus it will be considered as background.

3.4 Results and discussions

3.4.1 AP scores

The AP score results of object retrieval are presented in the Figure 5. The two set of results are from two different combinations of datasets and features: SS (small dataset with small feature, $L = 3$, $k = 5$, $cR = 200$, $K_{neighbor} = 10$), TT (tiny dataset with tiny feature, $L = 3$, $k = 5$, $cR = 150$, $K_{neighbor} = 10$.):

Category	AP Score SS	AP Score TT
_background	0.99353572	0.978499409
airplane	0.062378959	0.053752267
bear	0.061972371	N/A
bicycle	0.132069744	0.33711182
bird	0.081158264	0.390014869
boat	0.071705475	0.052094643
bus	0.11314982	0.272384312
car	0.107451717	0.283147066
cat	0.074707033	N/A
cow	0.107037698	0.052631579
dog	0.135864088	0.461988304
elephant	0.053829563	0.068016401
giraffe	0.065649489	0.052085792
horse	0.071286027	N/A
motorcycle	0.072824842	0.155474894
sheep	0.08417207	N/A
train	0.070730781	0.103090746
truck	0.127117587	0.097354858
zebra	0.078378172	0.052699349
mAP	**0.08730465**	**0.17370335**

Figure 5: AP scores of object retrieval by K-nearest neighbor structure.

3.4.2 Space-Quality tradeoff

The relationships between average distance to the K nearest neighbors vs search time and mAP vs the search time are shown in Figure 6. Here the data points are acquired by varying cR in a range of $[\infty, 400, 350, 300]$ while keeping $L = 3$, $k = 5$, $K_{neighbor} = 10$.

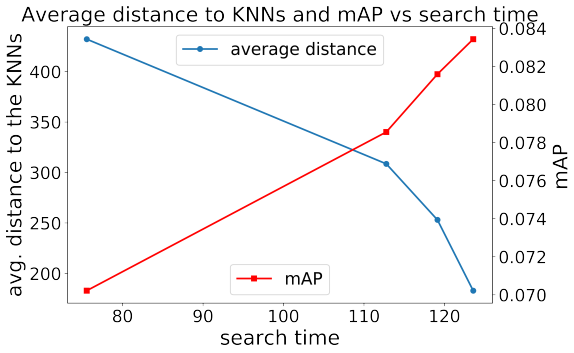

Figure 6: Tradeoff between the speed vs the quality of the search..

The results indicate that there is a tradeoff between search speed and the quality of the search, the tendencies of which are shown in Figure 6: Increasing the search time for a query point yields higher mAP score and smaller distance to the K-nearest neighbors, indicating better search quality but it will also requires longer time to perform searching.

3.4.3 Sparsity of K-nearest neighbors

The relationships of mAP vs $K_{neighbor}$ on SS (small dataset with small feature, $L = 3$, $k = 5$, $cR = 200$) and TT (tiny dataset with tiny feature, $L = 3$, $k = 5$, $cR = 150$) are shown in Figure 7, with varying $K_{neighbor}$ in a range of $[3, 5, 10, 15, 20, 30, 40, 50]$.

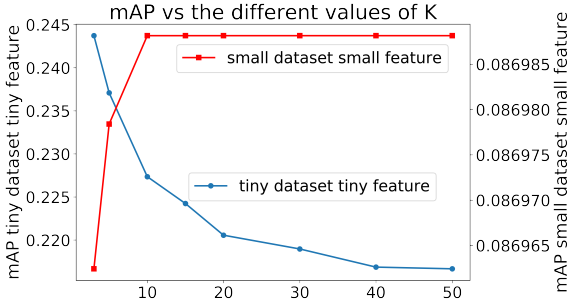

Figure 7: mAP vs $K_{neighbor}$ for different values of $K_{neighbor}$.

Ideally, the mAP should increase with increasing number of $K_{neighbor}$. However, here are two problems shown in Figure 7. For SS (red line), with increasing $K_{neighbor}$ the mAP increased initially and then reached a plateau after $K_{neighbor} = 10$, which means the neighbor distribution over buckets is too sparse or cR is too small, thus the query point had no more neighbors to find even with $K_{neighbor} > 10$, and the mAP stopped increasing. On the contrary, for TT (blue line), the neighbor distribution over buckets is too concentrated or cR is too large, thus the algorithm considered some points with large distance from the query point as its K-nearest neighbors, leading to decreasing mAP.

4 Conclusion

In this report, the approachs, algorithms, results, and analysis of object detection and retrieval by linear regression classifier and locality-sensitive hashing are discussed.

References

[1] Lin TY. et al. (2014) Microsoft COCO: Common Objects in Context. In: Fleet D., Pajdla T., Schiele B., Tuytelaars T. (eds) Computer Vision – ECCV 2014. ECCV 2014. Lecture Notes in Computer Science, vol 8693. Springer, Cham.

[2] Euclidean LSH, ML Wiki. http://mlwiki.org/index.php/Euclidean_LSH.