Large-scale Image Labeling via Mapreduce Topic Modeling

[Spring 2012 CMSC828G Final Project Report] *

Khoa Doan
University of Maryland
Dept. of Computer Science
College Park, MD 20742
khoadoan@cs.umd.edu

Rahmatri Mardiko
University of Maryland
Dept. of Computer Science
College Park, MD 20742
mardiko@cs.umd.edu

Ang Li
University of Maryland
Dept. of Computer Science
College Park, MD
angli@cs.umd.edu

ABSTRACT

Large scale computer vision is recently being popular due to the emerging of large scale imagery data and in need of more generally trained visual models. Recent advances in feature extraction provide a feasible and succinct way to represent image regions. However, most of these features are computationally heavy to extract. Since there is no explicit method to essentially speed up feature extraction, parallelization becomes the most comfortable strategy for this task. Due to the inherent high dimensionality of visual data, extracted features can be very noisy and thus not representative for images. Feature quantization is introduced to group similar features into the same low level semantics. K-means clustering is one of the usual choices for quantization while it may suffer from either time or space problems in ordinary environment. Latent Dirichlet Allocation is one of the techniques to discover higher level topic semantics for the context, which was popular in natural language processing. In this project, we unearth the potential of adapting computer vision tasks such as low level feature extraction and higher level image understanding into the MapReduce framework for large scale image dataset. We adopt the MapReduce LDA (Mr.LDA) to find topics across the images and discuss its possible extensions with respect to the image domain.

Keywords

Topic modeling, image labeling, parallelization, MapReduce framework

1. INTRODUCTION

Large scale visual data analysis and machine learning have been recently received much attention during the past few years. Traditional computer vision research focuses on small datasets of images or videos which makes the generalization of these methods difficult. In the current world of big data, a lot of imagery data have been present in the Internet. How to make use of the large scale of data to explore a better visual model has been one of the central problems in the current community. However, one natural problem arised from this task is heavy computational load. Computer vision tasks are usually involved with feature engineering which requires a lot of time and space for experiments. Fortunately, the MapReduce framework provides a reliable approach to large scale data processing.

In this work, we explore the potentials of using MapReduce framework for a standard computer vision task i.e. image labeling. The objective of the image labeling task is to find underlying semantics for each of the image pixels and to find groups of pixels that belong to the same object or semantic. Image labeling has been investigated for decades, although the scale of the dataset is limited due to computational issues. However, the demand of large scale image labeling turns out to be more and more clear in the recent years. One of the reasons is that people are being aware of using computer techniques to assist human annotation for specific imagery data such as remote sensing data. Millions of satellite images are generated for every day and the task of understanding these data is never feasible for human to do exhaustively. The automatic way will benefit the community tremendously in different areas such as surveillance, city planning, national defense, etc.

The rest of this paper is organized as follows. Section 2 briefly introduces the system structure of this project. MapReduce framework for visual feature extraction is discussed in section 3. Details of Mr.LDA is introduced in section 4. In section 6, a few implementation details are discussed. Section 7 shows the evaluation methods and experimental results for this project. Discussion on possible extensions from LDA to Spatial LDA is presented in section 8. Finally, the paper concludes in section 9.

2. SYSTEM OVERVIEW

We hadoopify the general pipeline for image understanding in our work, which can be divided into the five major components as follows. It worths emphasizing that all the five components are carefully designed to be mapreduce jobs.

1. Preprocessing: Converting from raw data to sequence image files in order for the reuse of image data;

2. Feature extraction: Extracting raw visual image features from sequence images;

*Data-Intensive Computing with MapReduce

3. Feature quantization: Building codebook using K-means clustering to quantize the raw features (or sampled raw features for extremely big data);

4. Topic modeling: Variational Inference based Latent Dirichlet Allocation for the quantized feature codes;

5. Postprocessing: Merge the code-topic mapping to pixel-code mapping so as to generate the final mapping from pixel locations to the topics.

The details of each step is illustrated in following content.

3. FEATURE EXTRACTION

Feature extraction often dominates the most computational resources in computer vision tasks. In this section, we introduce a MapReduce based framework to extract visual features efficiently which distributes the computation loads for raw feature extraction and scales up the construction of feature codebook via MapReduce K-means clustering.

3.1 Scale Invariant Feature Transform (SIFT)

In this work, the Scale Invariant Feature Transform (SIFT) [5] is adopted to describe local regions across the images. Generally, a SIFT descriptor represent a region bounded by a rectangular box centered at (x_c, y_c). The bounding box is uniformly divided into $K \times K$ parts. The gradients are computed in these sub-regions and the gradient magnitudes in N_b (the number of bins) discretized orientations are computed. A histogram is then constructed to represent the distribution of the gradient magnitudes. Therefore, for each region, the SIFT descriptor is a $K^2 N_b$ dimensional histogram vector.

One of the advantages of SIFT descriptors is its invariance to image rotation and translation, which makes this feature very popular among the computer vision community. We base our project on the MPI-CBG JavaSIFT library[2].

3.2 Dense Sampling SIFT Features

In order to apply SIFT features to represent the whole image for the task of image labeling, each of the pixels ideally should be described. Due to the fact that a neighborhood of pixels usually have little difference and belong to the same semantics, we sample series of small rectangular regions with a parameter STEPSIZE densely across each of the image and compute the SIFT description for each of the regions. The STEPSIZE controls the number of pixels between two consecutive centers of bounding boxes. Thereafter, each image can be converted from visual data to a list of (KEY,VALUE) pairs where the KEY is the index (g_i, x_i, y_i) of the regions and the VALUE is the SIFT feature vector f_i. g_i is the id of the image that contains the i-th region and (x_i, y_i) is the center location of the i-th region. Figure 1 illustrates how the dense sampled SIFT feature extraction works.

3.3 Image Input Aggregation

Given a large number of image files as input, there are several ways of loading them into a MapReduce job. One possible way is creating a text file containing a list of file names and let the mapper reads the files from HDFS and processes them while the reducer does nothing. An alternative way is similar to the first except the reading and processing is moved to the reducer while the mapper just passes the (key,value) pairs. Both are relatively simple and easy to implement. However, due to the small-files problem in Hadoop[7], neither the first and the second approach achieves the best performance in terms of processing speed.

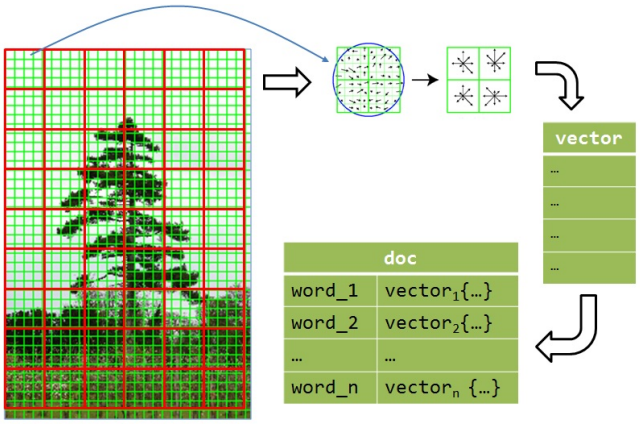


Figure 1: Illustration of dense SIFT feature extraction

Since Hadoop framework works best with large files, we need to aggregate the images in the dataset into a few big files. This step allows us to speed-up the feature extraction (and other images processing tasks) given a large number of image files as input. We adopt the approach presented in[8] which creates a MapReduce job that packs multiple files into a `SequenceFile`. This aggregation is performed as a preprocessing step for the other jobs that take images as input.

3.4 MapReduce-based Feature Extraction

We experimented two types of mapreduce based feature extraction strategies for image dataset (a) extracting features directly for unprocessed original image dataset and (b) extracting features for preprocessed sequentialized image files.

3.4.1 Feature extraction for unprocessed images

A list of image paths is generated in order to locate the image files. Each of the paths is assigned a number as image ID. The MapReduce job takes the path list as input and output sequence files containing region indices as keys and SIFT feature descriptors as values. Generally, the mappers distribute the paths into different reducers. Each of the reducers reads images from HDFS paths, extracts and emits the features.

3.4.2 Feature extraction for sequentialized images

One might find that reading data from HDFS does not sufficiently make use of the Hadoop architecture. Therefore, images can also be sequentialized to SequenceFiles. The key is the image id and the value is a byte array that encodes the corresponding image data. The preprocessing of sequentialization can be done aforehand. The details of conversion between origin image data and Sequence files will be explained in section 6.

Algorithm 1: Feature Extraction for Unprocessed Images

```
1  class Mapper
2      method map(LongWritable key, Text value)
3          (id, path) = split(value);
4          emit (id, path)
5  class Reducer
6      method reduce(IntWritable key, Text value)
7          id = key;
8          image = readImageFromHDFS(value);
9          feature = extractFeature(image);
10         emit (id, feature)
```

Algorithm 2: Feature Extraction for Sequentialized Images

```
1  class Mapper
2      method map(IntWritable key, BytesWritable value)
3          image = readImageFromBytes(value);
4          feature = extractFeature(image);
5          emit (key, feature);
```

3.4.3 *Evaluation and Comparison*

We compare these two feature extraction methods on a cluster using different sizes of image dataset. The two methods are evaluated under the same settings. Time consumption for either of the methods is shown in Fig.(2). The results show that reading data from HDFS can be expensive and sometimes unstable. The reason why the first method becomes expensive is because in Hadoop data is stored distributedly in different nodes and simply using HDFS paths as mapreduce job input do not give information about which node the specific image file is stored in. Therefore, cross-node data streaming may happen. However, the reading cross-node data only takes place when the demanded file is not located in node of the reduce job, which generally randomly occurs. This explains why the time consumption does not vary smoothly and stably when the size of image set changes. As we can see, if only one feature is extracted

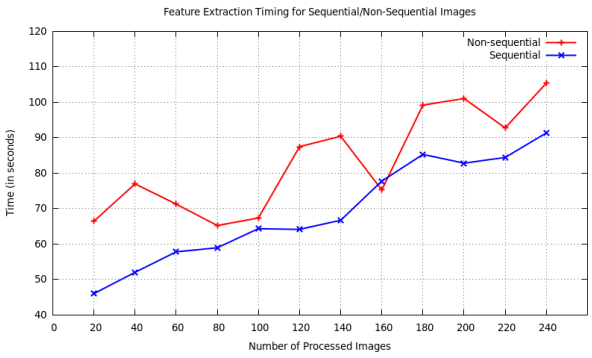


Figure 2: Time consumption for (a) reading non-sequential images from HDFS in mapreduce job, and (b) streaming Sequence Files in mapreduce job

at once, there should be no big difference between the two methods since the latter one still needs to read from HDFS to generate sequence image files. However, the major advantage of using Sequence files lies in the case that different types of visual features are needed to be extracted incrementally which can make our system more expandable. Even though the feature extraction is usually regarded as off-line processing, people might be interested in adding a new type of feature into the system or tuning different parameters of a specific feature type.

3.5 Building Feature Codebook

The raw features extracted from above generally have two problems in representing the images. On the one hand, the dimension of each feature vector is typically 128. For the similarity measure of each pair of small regions, at least 128 times of multiplications are necessary for Euclidean distances. This makes the further processing intractable even using parallelization framework. On the other hand, the feature vector is a histogram of oriented gradients which contains a lot of noises. Therefore, feature quantization is introduced to further processing the features and construct a "codebook" for the features. In the codebook, nearby feature vectors are grouped into the same index because they should belong to the same visual semantics.

K-means clustering is usually the choice for doing feature quantization. However, the clustering takes more features and thus needs much more memory spaces in order to exhaustively compute the distances between cluster centers to each of the points, as the scale of data becomes larger. The MapReduce framework generally resolve this time and space problem because not only the computation is distributed into different nodes but also the intermediate results of distances are stored almost uniformly among all of the nodes. In our project, we adopt Mahout K-means clustering [1] for building the codebook in Hadoop environment.

4. MAPREDUCE LDA

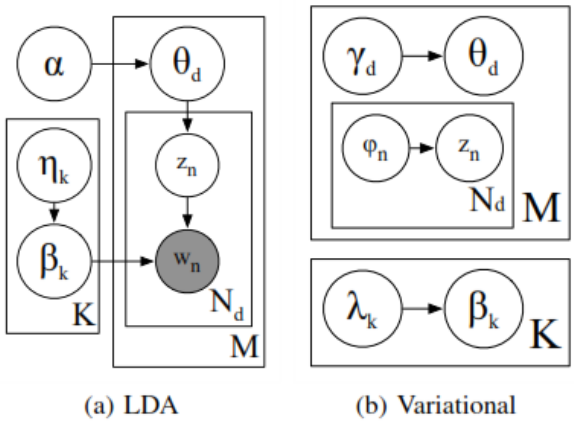


Figure 3: Graphical models for LDA

Mr.LDA is a MapReduce implementation of Latent Dirichlet Allocation (LDA) using variational inference [9]. LDA is a corpora topic model which clusters co-occurring words into topics. The graphical model of LDA is in fig.(3). LDA assumes the following generative prodedure of the corpus:

1. For each topic $k \in \{1, 2, ..., K\}$, a multinomial β_k is

sampled from $Dir(\eta)$.

2. For each document $m \in \{1, 2, ..., M\}$, a multinomial of topic distribution θ_m is sampled from $Dir(\alpha)$.
3. For each word n in a document m:
   (a) A topic label z_{mn} is sampled from a discrete distribution $Discrete(\theta_m)$.
   (b) The word $w_m n$ is sampled from a discrete distribution of topic $z_{mn}\tilde{\beta}_{z_{mn}}$.

The following section borrows heavily on [4] and [9]. On the high level, LDA computes the likelihood of the generating the the corpus $p(w|\alpha, \eta) = E_{p(\beta,\theta,z|\alpha,\eta)}p(w, \beta, \theta, z|\alpha, \eta)$. Since this is intractable to compute [4], Variational Inference or Monte-Carlos Markov-Chain (MCMC) sampling is used for approximation. MCMC techinique, such as Gibbs Sampling, is unbiased but undeterministic. In addition, it requires many iterations (in the order of thousands) to adequately walk the sample space, and synchonizatation of all variables at each iteration. For these reasons, Gibbs Sampling is not ideal for a MapReduce environment.

Variational inference, on the other hand, is deterministic procedure. For LDA, the variational distribution, as described in (3). Interestingly, maximizing the loglikelihood of the data with respect to the variational distribution results in updates of the variational parameters that can be efficiently distributed in a MapReduce environment, as described in [9]. Intuitively, document specific variables, $\phi_{v,k}$ and γ_m, can be computed in Mappers, and topic specific variables, λ_k, which is updated from aggregation of other document-specific variables for a specific topic k, can be computed in Reducers by shuffling all values for a specific topic to the same reducer.

In this paper, we bases our work on Mr.LDA. We have upgraded Mr.LDA implementation from MapReduce MRv1 environment to Cloudera MRv2 environment. In addition, one of the missing features of Mr.LDA is the ability to return topic labels for the documents in the corpus. This is essentially the output of the variational paramterers ϕ. The output of ϕ-table is written out in each Mapper (as described above) using MultipleOutput in MapReduce. The labels of each word in the document is determined by finding the value $\phi_{v,k}$ that is above some pretermine threshold. For our image-dataset, we set this value to 0.7.

5. MAPREDUCE SPATIAL LDA

Since LDA describes a document as a bag-of-word, direct application of LDA on images would cause several problems. In LDA, co-occurring words tend to cluster into the same topic even if these words are further away. For images, this is often not correct because some objects, such as buildings and streets, usually occur together in the same document but they are in fact different objects. This is because LDA does not take in to account the spatial constraint of the visual words, i.e. words that are close in space should be more likely from the same object.

Spatial LDA proposes a solution the above problems. In general, Spatial LDA consider an image as a collection of several documents. The probability of a visual word to belong to a document depends on its spatial distance to the document, thus it encourages grouping of visual words that are close in proximity (which are more likely to belong to the same object) into the same documents. The graphical model of Spatial LDA is in fig.(4) and its generative procedure is described as below:

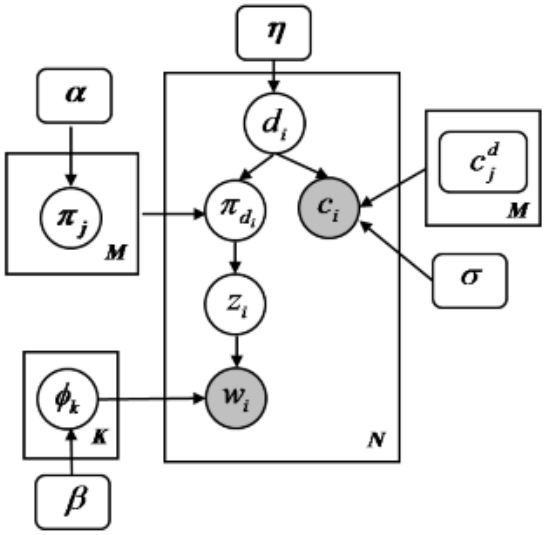


Figure 4: Graphical models for LDA

1. For each topic $k \in \{1, 2, ..., K\}$, a multinomial ϕ_k is sampled from $Dir(\beta)$.
2. For each document $m \in \{1, 2, ..., M\}$, a multinomial of topic distribution π_m is sampled from $Dir(\alpha)$.
3. For a visual word i:
   (a) A document assignment variable d_i is sampled from a uniform discrete distribution $Discrete(d_i|\eta)$.
   (b) Location c_i is sampled from a Gaussian kernel: $p(g_{c_i}|g_{d_i}, \sigma) = \delta_{g_{d_i}}(g_{c_i}) \exp -\frac{(x_{d_i}-x_i)^2+(y_{d_i}-y_i)^2}{\sigma^2}$, where (x_i, y_i) and (x_{d_i}, y_{d_i}) indicates the locations of word i and document d_i, respectively.
   (c) A topic label z_i is sampled from a discrete distribution of document d_i: $Discrete(\pi_{d_i})$
   (d) The word w_i is sampled from a discrete distribution of topic z_i: $Discrete(\phi_{z_i})$

[6] uses Gibbs Sampling as an approximation to the Spatial LDA inference problem. In this report, we propose the Variational Inference method in solving Spatial LDA. We expect this approach will allow us to extend Mr.LDA using Spatial LDA for our image dataset. The derivation of Spatial LDA is described in the Appendix.

6. IMPLEMENTATION REMARKS

In this section we present some implementation details that we did in the project.

6.1 Image-to-SequenceFile Conversion

In the evaluation phase of the project we need to compare the LDA output and the true labels. To enable pixel-by-pixel evaluation we extract each pixel in the ground truth images and store them as (key,value) pairs. The key, which contains the image id and the pixel position (x,y), is of type `TripleOfInts` whereas the value is of type `IntWritable` and it contains the pixel value. This images-to-pixels conversion can actually be performed as map-only MapReduce job. However, sometimes it is useful to group pixels that have the same semantic together. Here we can apply value-to-key conversion so the pixels that have the same labels are grouped together when they arrive at the reducer.

Algorithm 3: Conversion from Images to SequenceFiles

```
1  class Mapper
2      method map(IntWritable key, BytesWritable value)
3          image = readImageFromBytes(value);
4          (width, height) = sizeOf(image);
5          for y = 1 → height do
6              for x = 1 → width do
7                  pixel = getPixel(image, x, y);
8                  emit (pixel, triple(key, x, y));
9  class Reducer
10     method reduce(IntWritable key,
       Iterable<TripleOfInts> value)
11         iter = values.iterator();
12         while iter.hasNext() do
13             emit (iter.next(), key);
```

6.2 Sequencefile-to-Image Conversion

Also for the evaluation purpose, we need to build images from a set of pixels. This task is the inverse of the previous task. The mapper takes ((image_id,x,y),pixel) pairs and produce (image_id,(x,y,pixel)) as intermediate key value pairs so the pixels that belong to the same image are grouped together in the reducer. In the reducer the pixels are used to build the image object and convert it to `BytesWritable`. Since the output is in the form of `SequenceFile`, we still need to read the output separately to get the individual image files.

6.3 Sampling data for K-means clustering

The K-means clustering involves heavy computation on the Euclidean distances between cluster centers and each of the data point. Therefore, for extremely large dataset which might contain millions of regular images, the number of 128 dimensional feature points can exceed billions. The scale of data are very redundant and not able to be processed even in Hadoop mapreduce framework. However, for the building of feature codebook, we don't need to include all of the features in clustering. Instead, uniform sampling features usually gives a good estimate of the feature distribution and produces the codebook with same performance. In this project, for the large scale satellite image dataset, we use a mapper-only mapreduce job to random sampling data points. For each mapper, a random number is generated from 0 to 1 and if this number is less than a user-specified `ratio`, then the corresponding feature is sampled into the feature pool for clustering. Otherwise, the mapper does not output anything.

Algorithm 4: Conversion from SequenceFiles to Images

```
1  class Mapper
2      method map(TripleOfInts key, IntWritable value)
3          (id, x, y) = getMembers(key);
4          emit (id, triple(x, y, value));
5  class Reducer
6      method reduce(IntWritable key,
       Iterable<TripleOfInts> value)
7          image = new image(width,height);
8          iter = values.iterator();
9          while iter.hasNext() do
10             (x, y, pixel) = getMembers(value);
11             setPixel(image, x, y, pixel);
12         bytes = getBytes(image);
13         emit (key, bytes);
```

6.4 Merging pixel-code/code-topic mappings

The LDA will generate a mapping from the words in codebook to the set of topics. However, we want the results to be a mapping from pixel locations to the set of topics. In the feature extraction and codebook building steps, a big mapping from pixel locations to codebook has been constructed. Therefore, a connection operation between the two mapping is necessary to generate the final topic images. Fortunately, the codebook-topic mapping won't be large. Especially, the number of topics should be much less than the codebook size because topics represent a higher level of semantics in images. Therefore, it is affordable to load the codebook-topic mapping from HDFS in each of the reducers and join the two mappings in only one mapreduce job.

7. EXPERIMENTAL EVALUATION

7.1 Dataset

7.1.1 MSRC Image Labeling Dataset

The MSRC image labeling dataset (version 1) [3] contains 240 images and 9 object classes in total. Each of the images is pixel-wise labeled and contains 320×213 pixels. Fig.(5) shows some sample images from MSRC dataset.

Figure 5: Sample images from MSRC Image Labeling Dataset

7.1.2 Eastern Coast Satellite Image Dataset

We also collect satellite images using Google Maps APIs along the eastern coast in the United States. Each of the image is of 8-bit colormap PNG format and contains 402×415 pixels. Due to the capacity of our cluster, we pick 16,401 images amongst the dataset for the experiments. Fig.(6) shows some sample images from Satellite image dataset.

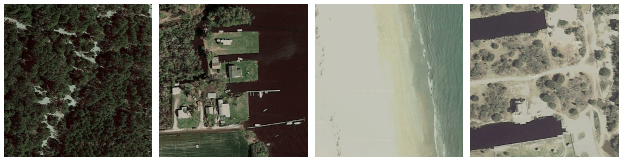
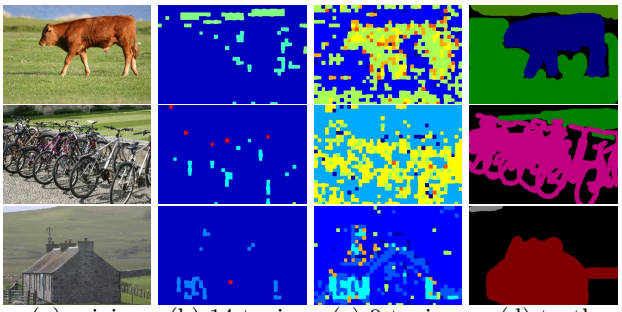


Figure 6: Sample images from Eastern Coast Satellite Image Dataset

7.2 Qualitative Evaluation

Images with labels in different colors are recovered from the output of LDA to show qualitatively which parts belong to the same semantics. Figure 7 presents three sample images and the output labels with 9 and 14 topics. The ground truth consists of 14 distinct semantics. However, the performance of LDA is significantly higher with 9 topics than 14 topics.



(a) origin (b) 14 topics (c) 9 topics (d) truth

Figure 7: Sample LDA output from MSRC dataset. The second and the third column show the output of labeling with LDA using 14 and 9 topics. The last column show the ground truth images.

7.3 Quantitative Evaluation

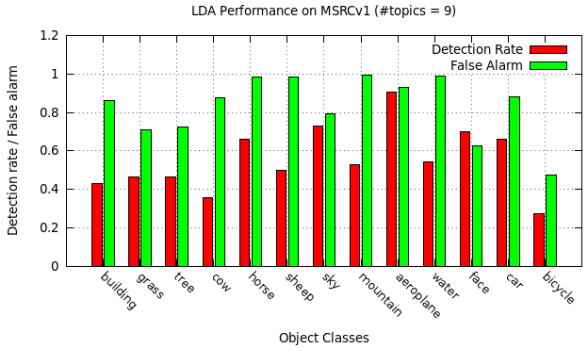
Traditional computer vision dataset has ground-truth for evaluation. For the MSRC image labeling data, each image is corresponded to a groundtruth image in which the whole image is divided into several components. Each component is annotated using a unique RGB color. In our output, each pixel is assigned to a type of semantics. Since our approach is completely unsupervised, the truth meaning of each semantic is not understandable. One possible way to evaluate the performance is to enumerate every possible correspondence between the two set of semantics. Apparently this strategy only works when the number of topics is small enough. It becomes difficult to construct a one-one correspondence between our labels and the groundtruth labels. In fact, how to accurately evaluate the performance of image segmentation tasks is still an open problem.

In this project, we propose to evaluate the results for each of the topics separately. For each semantic in the ground truth, a binary image can be constructed, regions with value 1 are those containing such semantic and otherwise not. For the output of the resulted topics, the topic that has the most number of occurrences in the semantic regions are chosen as a correspondence. The precision and recall are then computed according to that topic. Suppose the ground truth is $G : \mathbf{x}(id, x, y) \rightarrow \{1, 2, \ldots, K\}$ and the labeling map is $P : \mathbf{x}(id, x, y) \rightarrow \{1, 2, \ldots, K'\}$, then the evaluation score for each class k can be computed as

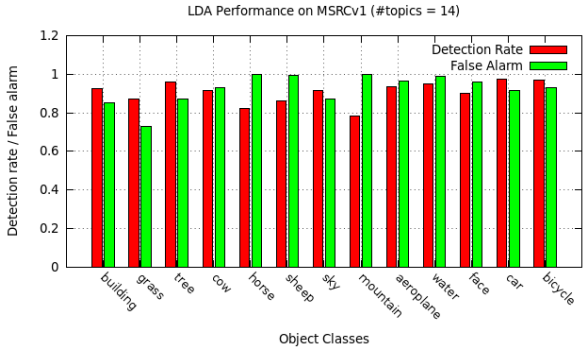
$$\text{DetectionRate}(k) = \frac{\text{cardinal}\left(\{\mathbf{x} \mid P(\mathbf{x}) = k\}\right)}{\text{cardinal}(\{\mathbf{x} \mid G(\mathbf{x}) = k\})} \quad (1)$$

$$\text{FalseAlarm}(k) = \frac{\text{cardinal}(\{\mathbf{x} \mid G(\mathbf{x}) \neq k\})}{\text{cardinal}(\{\mathbf{x} \mid P(\mathbf{x}) = k\})} \quad (2)$$

where cardinal$(\cdot)$ denotes the size of the set.



(a) 9 topics



(b) 14 topics

Figure 9: Quantitative evaluation for each classes

8. DISCUSSION: SPATIAL LATENT DIRICHLET ALLOCATION

9. CONCLUSION

Acknowledgments

The authors would like to thank Prof. Jordan Boyd-Graber and Prof. Jimmy Lin for their consistent help in this project.

10. REFERENCES

[1] Mahout: http://mahout.apache.org/.

[2] mpi-cbg javasift library: http://fly.mpi-cbg.de/ saalfeld/projects/javasift.html.

[3] Msrc dataset: http://research.microsoft.com/en-us/projects/objectclassrecognition/.

[4] D. M. Blei, A. Y. Ng, M. I. Jordan, and J. Lafferty. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:2003, 2003.

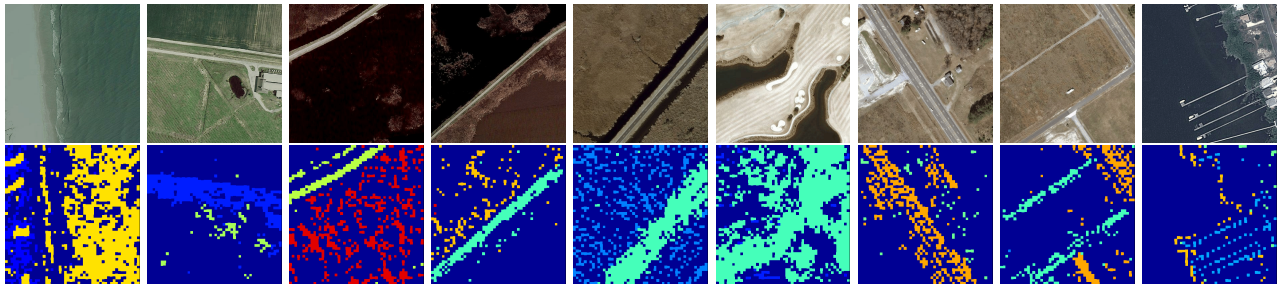


Figure 8: Qualitative results for Google Maps Satellite Image Dataset

[5] D. G. Lowe. Distinctive image features from Scale-Invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, Nov. 2004.

[6] X. Wang and E. Grimson. Spatial latent dirichlet allocation. In *NIPS*, 2007.

[7] T. White. The small files problem: http://blog.cloudera.com/blog/2009/02/the-small-files-problem/.

[8] T. White. *Hadoop: The Definitive Guide*, pages 239–245. O'Reilly Media, Inc., 1st edition, 2009.

[9] K. Zhai, J. B. Graber, N. Asadi, and M. L. Alkhouja. Mr. LDA: a flexible large scale topic modeling package using variational inference in MapReduce. In *Proceedings of the 21st international conference on World Wide Web*, WWW '12, pages 879–888, New York, NY, USA, 2012. ACM.

APPENDIX