

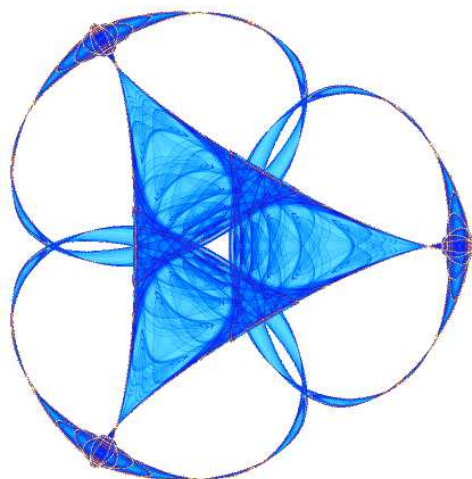
VISUAL WORDS, TEXT ANALYSIS CONCEPTS FOR COMPUTER VISION

By

**Wang-Juh Chen, Hoi Tin Kong, Minah Oh,
Patrick Sanan, Ying Wang,
and
Brendt Wohlberg**

IMA Preprint Series # 2287

(November 2009)



INSTITUTE FOR MATHEMATICS AND ITS APPLICATIONS

UNIVERSITY OF MINNESOTA
400 Lind Hall
207 Church Street S.E.
Minneapolis, Minnesota 55455-0436

Phone: 612/624-6066 Fax: 612/626-7370

URL: <http://www.ima.umn.edu>

Team 6 Final Report: Visual Words, Text Analysis Concepts for Computer Vision

Wang-Juh Chen* Hoi Tin Kong[†] Minah Oh[‡] Patrick Sanan[§]
Ying Wang[¶]

Mentor: Brendt Wohlberg^{||}

August 27, 2009

Abstract

In this project, we introduce some new algorithms that build an Image Annotation Engine via machine learning. We also consider new methods to improve such algorithms. Numerical experiments that provide empirical support to the theory is provided.

*Department of Mathematics, Arizona State University, wangjuhchen@asu.edu

[†]Department of Mathematics, University of Georgia, kong@math.uga.edu

[‡]Department of Mathematics, University of Florida, oh@ufl.edu

[§]ACM, California Institute of Technology, psanan@acm.caltech.edu

[¶]Department of Mathematics, Ohio State University, wang@math.ohio-state.edu

^{||}Los Alamos National Laboratory

1 Introduction

While significant progress has been made in information retrieval in text documents, tools for indexing multimedia data are still primitive. One of the most difficult but important component of such system is Automatic Image Annotation. Automatic Image Annotation is the process of the computer system automatically assigning keywords to a given digital image, and this is used in image retrieval systems to locate images that are required. In this project, we will build an Image Annotation Engine via ‘machine learning’, which are algorithms that train computers to recognize certain features using the database. In the next section, we will learn about the algorithm Scale Invariant Feature Transform (or SIFT) that selects and characterizes local features in a given image. In Section 3, a simple ad hoc algorithm will be introduced, and the effectiveness of this algorithm will be illustrated through numerical experiments. In Section 4, we explain the notion of clustering in detail and suggest new methods that can improve or avoid clustering with the potential of getting better results. In the following two sections, new algorithms that are mathematically more justifiable than the ad hoc algorithm are introduced. Finally, we conclude this report with a summary and future work section.

2 Scale Invariant Feature Transform (SIFT) and Experiments

2.1 SIFT

Scale Invariant Feature Transform (SIFT) [4] is a method to extract distinctive features from an image which is invariant under scaling, rotation, addition of noise, and illumination change etc. This method was proposed by David Lowe in 1999 and has been applied to many areas including object recognition, robotic mapping and navigation, image stitching, 3D modeling, gesture recognition, video tracking, and match moving [6].

The first step of SIFT is to find the key points of the image. The basic idea of extracting the keypoints is to apply Gaussian filters at different scales of the image. Then the extrema points are taken from the difference of successive Gaussian-blurred (Difference of Gaussians - DoG), and this is what we call keypoints. Given an image, $I(x, y)$, and a variable-scale Gaussian, $G(x, y, \sigma)$, the scale-space of an image is defined as [4]

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y), \quad (2.1)$$

where $*$ is the convolution operation in x and y , and

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2}, \quad (2.2)$$

and the Difference of Gaussian function can be calculated by

$$\begin{aligned} G(x, y, \sigma) &= (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y) \\ &= L(x, y, k\sigma) - L(x, y, \sigma), \end{aligned} \quad (2.3)$$

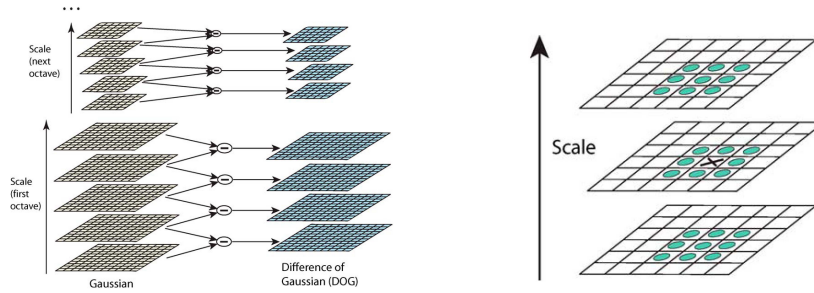


Figure 1: **(Left)** Difference of Gaussian (DoG) is calculated by repeatedly convolving the image with Gaussians to produce the set of scale space images and **(Right)** the adjacent Gaussian images are subtracted to produce the DoG. (Source [4])

where k is a constant multiplicative factor which separates two nearby scales. Fig 1 illustrates how this algorithm works.

The keypoints acquired from DoG are the candidate locations, and a detail model will be applied to determine how stable, in the statistical sense, a keypoint is. Only the stable keypoints will be considered as the real features. Besides the invariant magnitude and the location of the features, SIFT also provided the orientation of each feature based on the local image gradient direction. But in our project, only the magnitude will be used to build the Image Annotation Engine.

2.2 Experiments

	Training		Testing	
	Images	Objects	Images	Objects
Bicycle	127	161	268	326
Bus	93	118	180	233
Car	271	427	544	854
Cat	192	214	388	429
Cow	102	156	197	315
Dog	189	211	370	423
Horse	129	164	254	324
Motorbike	118	138	234	274
Person	319	577	675	1153
Sheep	119	211	238	422
Total	1277	2377	2686	4753

Table 2.1: Summary of VOC2006 database.



Figure 2: Some example images from VOC2006 database.

2.2.1 Data set

We used the public image data sets from VOC2006 database [2] including predefined training and testing sets. The training set, which is used for training the image annotation machine, includes 1277 images and the testing set includes 2686 images. Table 2.1 summarizes statistics of the training and testing data. Notice that the original database also includes validation sets which does not apply in our algorithm, so the training and testing data sets mention in this project are those without the validation images. For each image, labels of the image are given in a individual annotation file with quantity and a bounding box which denotes the location of each object. Notice that the bounding box information is not used throughout the experiment. One reason of not using the bounding box information is because most of the images from internet do not provide this information and we do not want our algorithm limited on this special type of images. We also would like to see how robust our algorithm is with lots of features not coming from the specific objects (outside of the bounding box). There are 10 object classes(labels): bicycle, bus, car, cat, cow, dog, horse, motorbike, person, sheep. Every image has of one or more labels and the examples images are shown in Figure 2.

2.2.2 Experiment setup

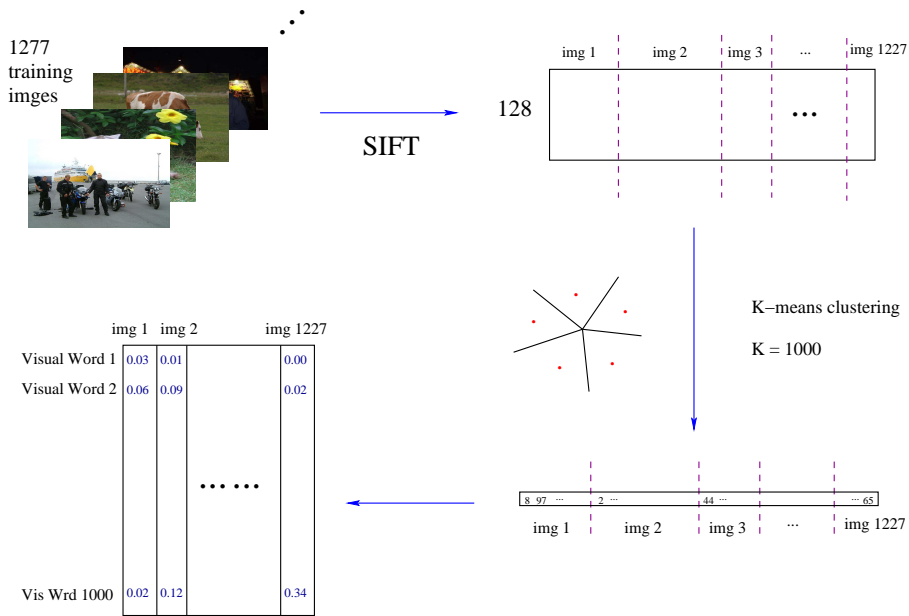


Figure 3: The flowchart of the proposed algorithm.

Figure 3 shows the experiment setup in training stage. For each training image, we calculate the feature vectors of the image using the SIFT algorithm. Each feature vector is of length 128, and the number of the feature vectors for each image varies.

3 An Ad Hoc Algorithm and its Improvement

In this section, we will introduce an ad hoc algorithm to build an image annotation engine. The simplicity of this algorithm doesn't reduce the effectiveness, which can be illustrated by the numerical testing results. We will also show a few improvements of this algorithm.

3.1 An Ad Hoc Algorithm

The algorithm consists of 3 steps, and the 4th step is the assessment.

- Step 1: Each training image is assigned a visual word distribution
- Step 2: Each label learns its visual word distribution
- Step 3: Based on the visual word distribution, assign each testing image label(s)
- Step 4: Evaluation of the ad hoc algorithm

Each step will be explained in detail in this section.

Step 1: Each training image is assigned a visual word distribution

For every image, the SIFT algorithm reads in the grey-scale intensity matrix and gives a collection of feature vectors. The bag of feature vectors collected from all the training images is then sent to the k-mean algorithm. The k-mean algorithm clusters all the feature vectors into k clusters. In our case, $k = 1000$ is chosen. Now, among the feature vectors belonging to one training image, we can have a count of feature vectors in each cluster. Each cluster can be conceptually understood as a visual word. Therefore, for each training image, there's a visual word distribution associated to this image. In fact, this visual word distribution is THE representation of the content of this image. In addition to the visual word distribution, there's also a collection of tags assigned to this image.

Step 2: Each label learns its visual word distribution

Once the visual word distributions for all the training images are built, it's ready for the labels to learn their visual word distributions. The way this is done is as follows: since the labels want to learn their visual word distribution from all the training images, each training image is introduced (or shown) to all the labels.

For example, before the first training image is introduced to all the labels, all the labels have no knowledge about their visual word distributions, hence they all have blank visual word distribution. When the first image, which has labels bus (label 2) and car (label 3) (see Figure 4), is shown to all the labels, every label takes a look at the first image. The reaction of each label is in general different. For example, label 1 finds that she is not a label of the first image, so, label 1 thinks that the visual word distribution of the first image has nothing to do with her, hence, she will ignore this image, so do label 4-10. But, label 2 finds that she is one of the labels of the first image. She thinks that the visual word distribution of the first image should have similarity to her own visual

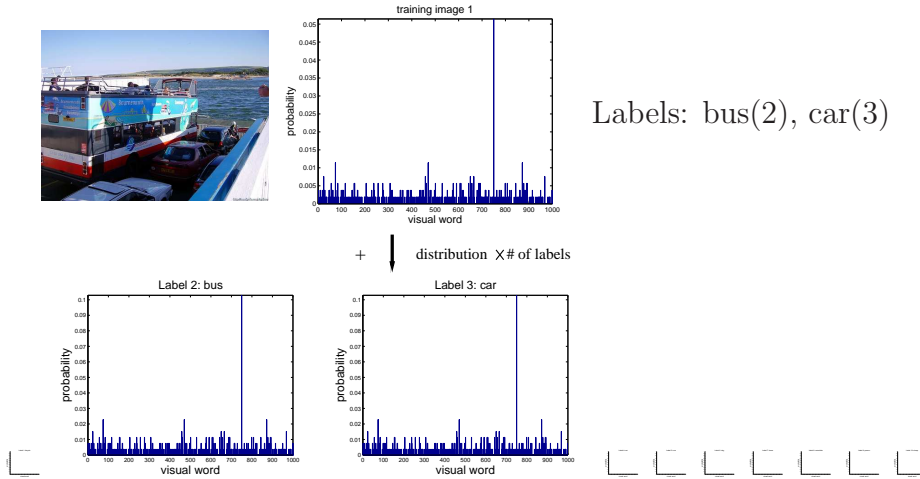


Figure 4: The labels learn their visual word distribution from the first training image.

word distribution. But there's no scientific justification about the form of the similarity, so, label 2 will simply get a copy of the visual word distribution of the first image and multiply it by the number of labels attached to the first image and add it to her empty visual word distribution. The reason we multiply the visual word distribution of the first image by the number of labels attached to the first image, rather than only add one copy of the visual word distribution of the first image to the blank distribution of label 2 is the following: the visual words appeared in the 2 labels of the first image split the distribution, hence the visual words appeared in label 2 only get half of the weight of what they should have got if there's only one label. Therefore, to restore the correct weight for each word appeared in label 2, we multiply the visual word distribution of image 1 by 2. Similarly, label 3 will do the same thing as label 2, because label 3 is also one of the labels for the first image. This training step is shown in Figure 4.

After all the labels have learnt from the first training image, the second training image will be shown to all the labels (see Figure 5). The second training image has 2 labels: label 1 and label 9. Hence its visual word distribution will be multiplied by 2 and added to the visual word distribution of label 1 and label 9 (see Figure 5).

This step will be repeated for every training image. Once the labels learnt their visual word distribution from all the training images, each of the label will have a visual word distribution on their own, for example, the visual word distribution of label 1 is shown in Figure 6. Heuristically, Figure 6 shows the probability that each visual word is associated to label 1, and it can also be conceptually understood as the probability that label 1 should be assigned when each visual word appears in an image, i.e. $P(l_1 \text{ should be assigned when } w_i \text{ appears in an image})$.

What can be seen from Figure 6 is the fluctuation at the bottom, and the fluctuation could be noise. To get rid of the fluctuation, we can impose a filtering parameter, call it θ , and everything below $\theta \times \max$ will be filtered out from the distribution as shown in Figure 7. Note that the parameter θ give one degree of freedom of our algorithm, we can tune its value for the better performance.

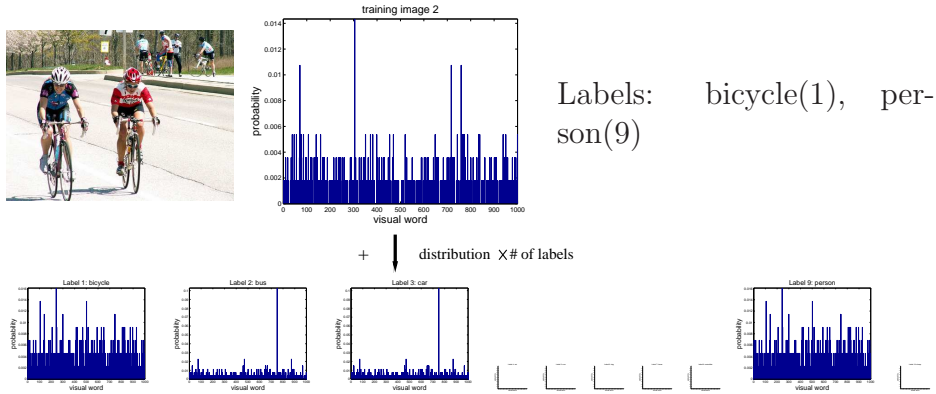


Figure 5: The labels learn their visual word distribution from the second training image.

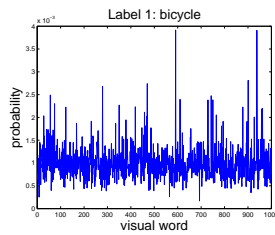


Figure 6: The visual word distribution of label 1.

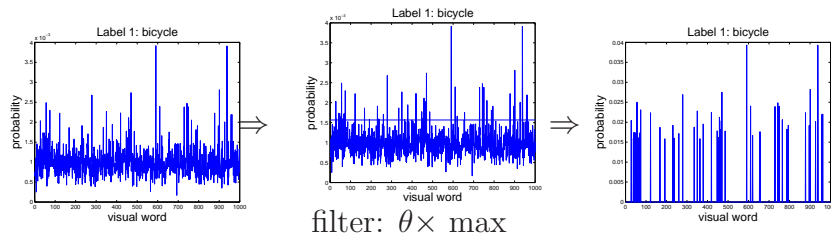


Figure 7: The process to filter out the fluctuation appeared in the visual word distribution of label 1.

Now, each label has its own visual word distribution, and if we collect them together for all the labels, we get a matrix as shown in Figure 8. Each column of this matrix represents the visual word distribution for each label. Heuristically, the (i, j) -entry means $P(l_j \text{ should be assigned when } w_i \text{ appears in an image})$.

Step 3: Based on the visual word distribution, assign each testing image label(s) When a testing image is given to the annotation engine, the engine first compute its visual word distribution as what's done to each training image. Then this row vector containing the visual word distribution of this testing image is multiplied by the visual word distribution matrix obtained in the previous step. Figure 9 shows the details. After the multiplication, we get a new row vector. The j -the entry gives the confidence that

	l1	l2	...	l10
w1	0.04	0.03		0.13
w1	0.00	0.01		0.03
⋮			⋮	
w1000	0.03	0.06		0.08

Figure 8: The visual word distribution matrix for all the labels.

label j should be assigned to this testing image, and the reason is the following: $P(l_j \text{ is assigned}) = \sum_{i=1}^{1000} P(w_i) \cdot P(l_j \text{ is assigned when } w_i \text{ appears})$.

		l1	l2	...	l10	
New image	w1	0.04	0.03		0.13	Label = (l1, l2, ..., l10)
	w1	0.00	0.01		0.03	
	⋮			⋮		
	w1000	0.03	0.06		0.08	

Figure 9: The visual word distribution matrix for all the labels.

After the confidence is calculated for each label, we will use a decision making threshold to help us to give the final labeling decision: $l_j \geq \max_{i=1..10}(l_i) \times \text{threshold} \Rightarrow l_j$ will be assigned. Note that the decision making threshold gives the second degree of freedom. We will vary this parameter to evaluate the effectiveness of our algorithm at the next step.

Step 4: Evaluation of the ad hoc algorithm

ROC curve will be used to evaluate the effectiveness of the algorithm. ROC curve is short form for Receiver Operating Characteristic curve, which is a plot of the true positive rate against the false positive rate for the different possible cutpoints of a diagnostic test. Figure 10 gives a demonstration of the ROC curve. Certainly high true positive rate and low false positive rate are desired and hence the upper left corner gives the perfect result. We'd wish the ROC drawn by varying the decision making threshold parameter of our algorithm will be as close to the upper left corner as possible.

We fix the filtering parameter $\theta = 0$, and vary the decision making threshold to draw the ROC curve, and the result is given in Figure 11.

For most of the labels, maybe except label 9 – person, this simple heuristic annotation engine is doing very well on the training images and reasonably well on the testing images.

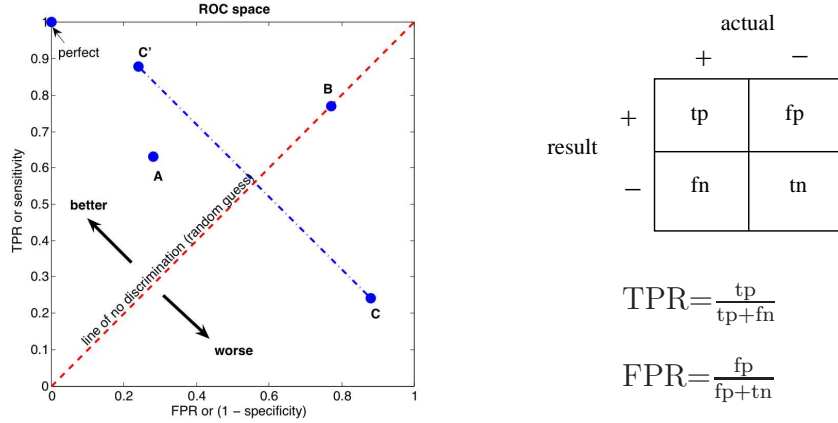


Figure 10: ROC space and true-false table

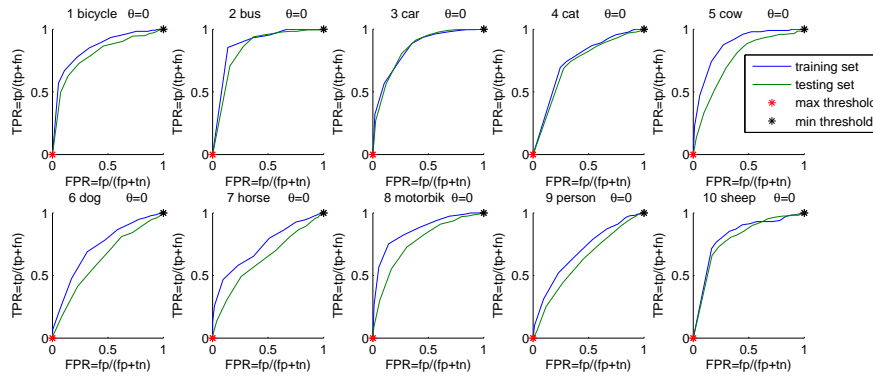


Figure 11: ROC curve for $\theta = 0.0$

3.2 Improvement to the ad hoc algorithm

Based on the elementary algorithm introduced in the previous section, we can make a few improvements through decision making step. Two of these are given in this section as examples of the possible improvement along this line.

3.2.1 Improvement through decision making – version 1

When a testing image is given to our image annotation engine, a confidence that each label should be assigned to this image is calculated, namely l_j . Previously, the way we make the final labeling decision is:

if $l_j \geq \text{threshold} \times \max_{i=1}^{10} l_i$, then l_j is assigned to this image.

To improve the performance, we can use the following decision:

if $l_j \geq \min_{i=1}^{10} l_i + \text{threshold} \times (\max_{i=1}^{10} l_i - \min_{i=1}^{10} l_i)$, then l_j is assigned.

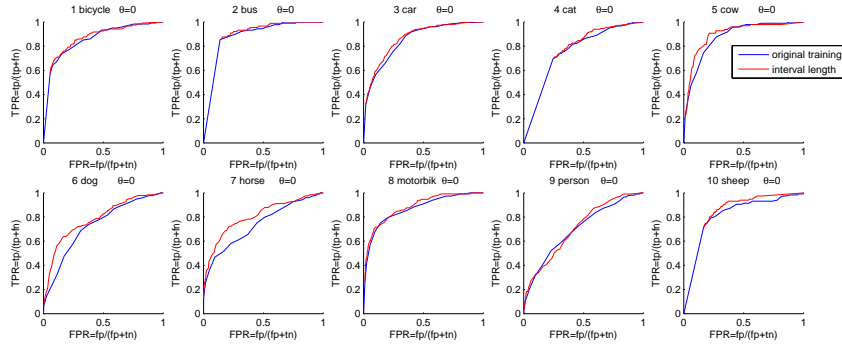


Figure 12: ROC curve for training images (improvement version 1)

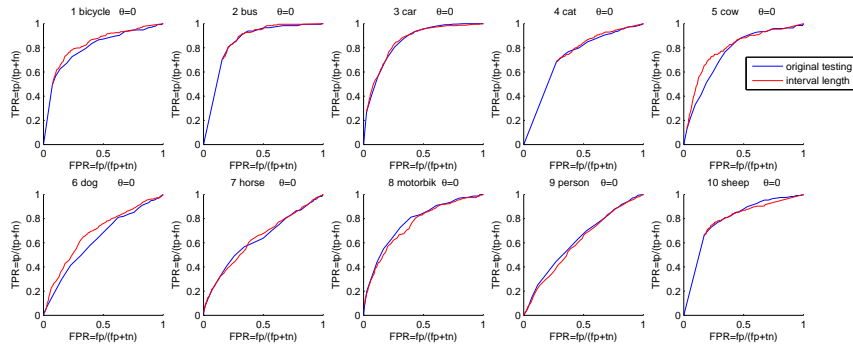


Figure 13: ROC curve for testing images (improvement version 1)

The ROC curve for the training images and testing images with this improvement are given in Figures 12 and 13 respectively. The original decision making is a special case of this improvement, in the sense that the $\min_{i=1}^{10} l_i$ was blindly treated to be 0. Figure 12 shows that improvement 1 does a lot better for labels 5,6,7 on the training images, which were originally relatively poorly performed. Whereas on the testing images, Figure 13 shows that improvement 1 does perform better for labels 5,6 on the testing images.

3.2.2 Improvement through decision making – version 2

Another way to make the final labeling decision is to compare the visual word distribution of the testing image with that of a single label, or combined labels. This means that we assume an image consists of 1 or 2 labels, then we build the ideal visual word distribution for such images, and then compare the given testing image with the ideal visual word distribution of images with one label or two labels. The decision making step

is:

if $\text{dist}(\text{testing image}, l_j) < \min(\text{dist}) + \text{threshold} \times (\max(\text{dist}) - \min(\text{dist}))$
then l_j is assigned to this image
if $\text{dist}(\text{testing image}, \{l_j, l_k\}) < \min(\text{dist}) + \text{threshold} \times (\max(\text{dist}) - \min(\text{dist}))$
then l_j, l_k are assigned to this image

The ROC curve for the training images and testing images with this improvement along with improvement 1 are given in Figures 14 and 15 respectively. Figures 14 shows that improvement 2 overperforms for labels 2,3,4,6,8,9 for the training images. 15 shows that improvement 2 overperforms for labels 2,4,7,8,9.

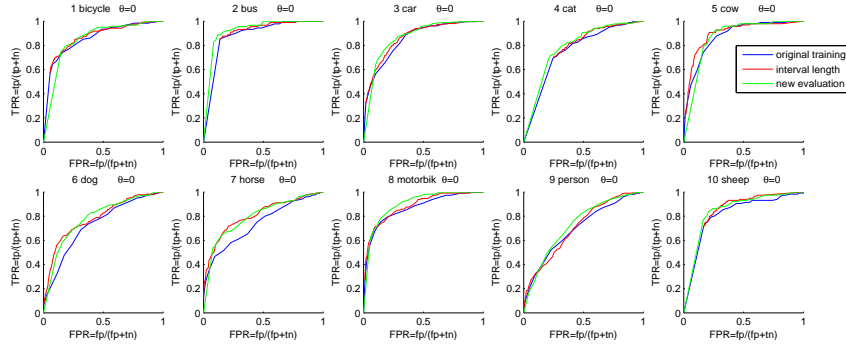


Figure 14: ROC curve for training images (improvement version 2: green, improvement 1: red)

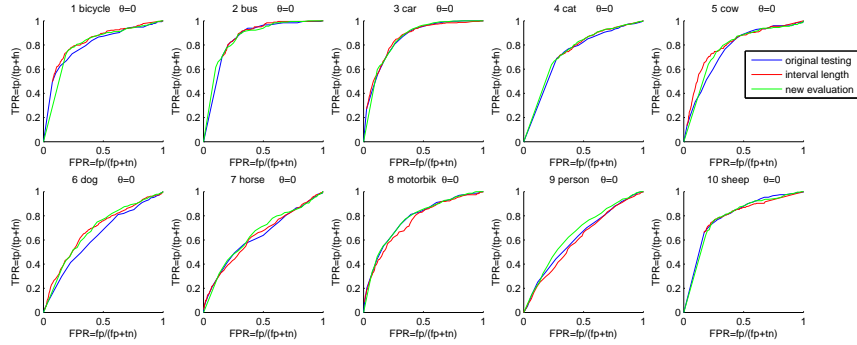


Figure 15: ROC curve for testing images (improvement version 2: green, improvement 1: red)

The two improvements shown here only serve as examples, there are numerous ways to improve the base-line ad hoc algorithm given in the previous section.

4 Cluster Analysis

In Section 2, we obtained a set of feature vectors using SIFT. In this section, we will consider methods that divide these feature vectors into subgroups in a way that all feature vectors in the same subgroup are ‘similar’ in a sense. The main idea of all methods presented in this section is to group the feature vectors that are close to one another in the vector space \mathbb{R}^{128} with respect to the usual Euclidean metric.

4.1 Clustering

Clustering is the division of a set into subsets (called clusters) so that elements in the same cluster are similar in some sense. Among many different types of clustering, in this project, we used the k -means Clustering. The first step in k -means Clustering is to divide the space \mathbb{R}^{128} into k clusters. The cluster centers, say $\mathbf{c}_i, 1 \leq i \leq K$, are the centroid of each cluster. Once this is done, each feature vector is assigned to one and only one cluster center that is in the nearest distance with respect to the Euclidean metric in \mathbb{R}^{128} . In other words, the feature vectors that are assigned to the same cluster center will be in the same subgroup so that after clustering we are left with k disjoint subgroups of feature vectors. In all numerical experiments in this project, we set $k = 1,000$, and it appears that this number is large enough to give promising results.

4.2 Improving Clustering

Although k -means clustering gave us good results as reported in the previous section, there are potential problems as illustrated in Figure 16.

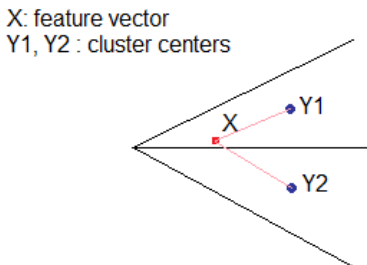


Figure 16: potential problems with k -means clustering

k -means clustering assigns one and only one cluster center to each feature vector. As illustrated in Figure 16, however, if the feature vector X is nearby the boundary of the cluster, X is almost as close to $Y2$ as it is to $Y1$. In this case, however, X is only associated with $Y1$ and the similarity between X and $Y2$ is ignored. In order to solve such potential problem, we considered a method that will improve the clustering in the following way.

For a fixed feature vector X , we first compute the distance between X and all cluster centers $Y_i, 1 \leq i \leq K$, and depending on these distances we give a weight to each cluster center. In particular, let d_i denote the distance from X to cluster center Y_i . Then, X is

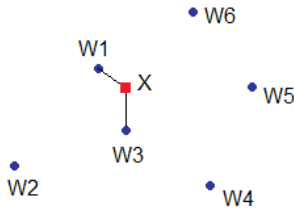


Figure 17: k -nearest neighborhood graphs

Table 4.1: 2-nearest Neighbors Graph Example

	$L1$	$L2$	$L3$	$L4$	$L5$	$L6$
X	$e^{-\alpha d_1}$	0	$e^{-\alpha d_3}$	0	0	0

associated with a length k vector where the i -th entry is $e^{-\alpha d_i}/C$, where α is a parameter and C is the constant that normalizes the vector. In contrast, k -means clustering will associate to each feature vector a length k vector with one on one entry and zero elsewhere. Although this idea seems promising, numerical experiments show that the improvement from this method is negligible. One of the reasons why the improvement was not noticeable may be that the clustering that we used was quite good. Since algorithms presented in this report will give different results depending on the clustering, this improvement method may be useful in the case when k -means clustering is giving poor results.

4.3 Avoiding Clustering

We also considered methods that avoid clustering. The following methods use the whole set of feature vectors instead of cluster centers, and computing the distance to/between all feature vectors is very expensive. This difficulty, however, can be overcome by using fast algorithms such as k -d tree [3] or Approximate Nearest Neighbors methods [5].

4.3.1 k -nearest neighborhood graphs

Given a set of feature vectors in a testing image, for a fixed feature vector X , we can compute the distance to each feature vector in the training set. This method chooses the nearest k feature vectors in the training set. Then each feature vector gets a weight depending on its distance from X .

Figure 17 and Table 4.1 is an example of how this method works when $k = 2$. X is a feature vector of the testing image, and $W_i, 1 \leq i \leq 6$ are the feature vectors of the training data. L_i is the label associated with W_i and d_i is the distance from X to W_i . Once we obtain a vector like Table 4.1 for each feature vector of the testing image, we normalize each vector and sum it up. By normalizing the resulting vector, we get the probability distribution of labels of the testing image.

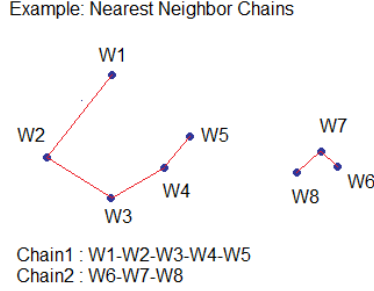


Figure 18: k -nearest neighborhood graphs

Unfortunately, this method did not work well. One reason of such failure could be that a large portion of the feature vectors in each training image did not represent the label attached to the image. For example, if we have a training image with the label ‘person’, a large portion of the feature vectors is coming from the background instead of the person. Since this method depends strongly on the accuracy of the feature vectors in the training image, in order to use this method, we believe that we need much cleaner data.

4.3.2 Chains

For this method, we first compute the distance between each feature vector in the training image. Then we choose one feature vector and start making a chain by attaching the nearest feature vector. The chain is completed once we have an infinite loop. When one chain is complete, we choose any other feature vector that is not in any chain and continue this process. We will eventually have a ‘chain decomposition’ of the feature vector space. Figure 18 is a simple example of such chain decomposition. Note that such chain decomposition is not unique as it depends on the choice of feature vectors that starts a chain.

For each chain, we take the intersection of the labels that are attached to the feature vectors in the chain to get a distribution of labels.

4.3.3 Support Vector Machine

Support Vector Machine, a very powerful classifier which can prevent the curse of dimensionality, is also considered in this project. The basic idea of SVM is to find a decision function (classifier), $f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b$, with maximum margin, given a training set, $\{\mathbf{x}^m, \mathbf{t}^m\}_{m=1}^M$ where $\mathbf{x}^m \in \mathcal{R}^N$ with class label $\mathbf{t}^m \in \{-1, +1\}$. The problem is then formulated by an optimization problem (primal)

$$\{\mathbf{w}^*, b^*\} = \operatorname{argmin}\left\{\min_{\mathbf{w}} J_{\text{H}}(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|_2^2, \right. \quad (4.1)$$

subject to $t^m[\mathbf{w}^T \mathbf{x}^m + b] \geq 1, m = 1, \dots, M\}$,

and the corresponding dual problem, a convex QP problem,

$$\alpha^* = \operatorname{argmin}_{\alpha} \left\{ \min_{\alpha} \left\{ \frac{1}{2} \sum_{n=1}^N \sum_{m=1}^M \alpha^n \alpha^m t^n t^m (\mathbf{x}^m)^T \mathbf{x}^n - \mathbf{e}^T \alpha \right\} \right. \\ \left. \text{subject to } \alpha^T \mathbf{t} = 0 \text{ and } \alpha \geq 0 \right\}, \quad (4.2)$$

where the vector α is the Lagrange multiplier. The advantages of SVM is the flexibility of choosing the kernel functions, the reduction of dimension by solving dual problem, and the usage of sparse solution (Support Vectors) in predicting (testing) stage.

We proposed two ways of using SVM. The first way is to treat images as samples, of dimension 127, and visual words as features, of dimension 1000, because SVM can only deal with binary situation decision, we have to train 10 SVM classifiers in our experiment. Take **car** as example, the class label \mathbf{t}^m is either 1 or -1 depended on whether **car** appears in that particular image or not. In this case, we do not take the advantage of solving the dual problem because the number of primal and dual variables are similar. The second way is to treat keypoints as samples, of dimension approximate 1 Million, and the dimension of the keypoint as features, of dimension 128. This also does not take the advantage of solving the dual problem, actually, it is even worse because now the dual variables are the dimension of 1 Million. The other problem of this is that not all the keypoints within one image are associated to the same label, in other words, we do not give specific labels for each feature vector when we train SVMs. Given lots of **incorrect** labels in training set, we should expect a poor prediction in the testing set. In general, we also suffered from lack of time to choose the parameters, i.e. the parameter to trade-off between the size of the margin and the total classification error, which is very import in SVM and ended up with unsatisfactory result. The geometry of SVM is shown in Figure 19.

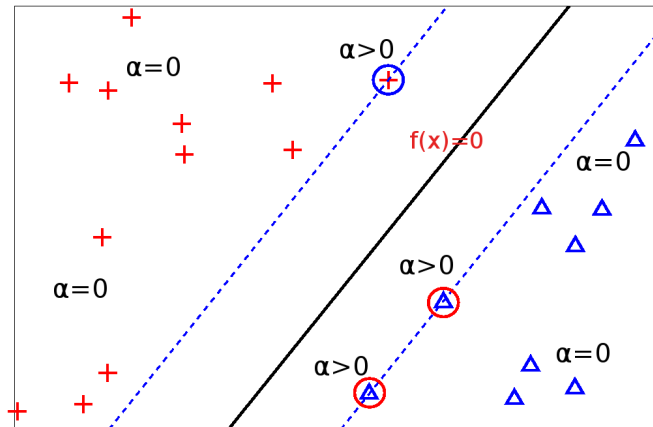


Figure 19: Support vectors (circled) are the data points lying on the margin corresponding to positive α .

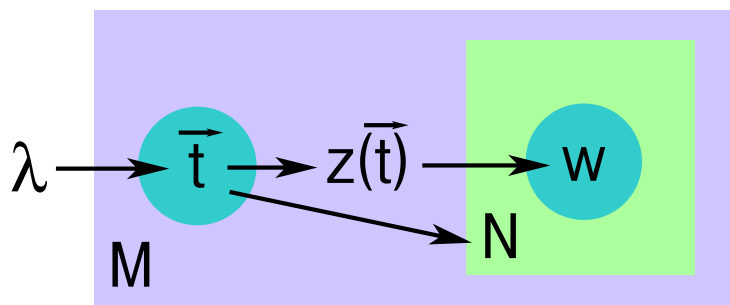


Figure 20: Plate Diagram for Simple Generative Model

5 A Simple Generative Model

A generative model is introduced and used to design a classifier. In particular, this model seeks to introduce a more justifiable generative model than our ad hoc approach, and ends up lending some justification to the ad hoc approach.

5.1 Generation Algorithm and Conditional Probabilities

Given a training set of images, each tagged with one or more from a set of L tags and associated with a vector of words \vec{w} , each of which take values amongst K cluster centers ('visual words'), we assume that each image/document (as characterized by its set of visual words and assigned tags) is generated by the following process:

1. From a multinomial distribution, choose a set of tags $\vec{t} \in \{0, 1\}^L$
2. Using some (deterministic) formula, associate a stochastic/probably vector $\vec{z}(\vec{t}) \in [0, 1]^K$
3. Choose a number of word N using some distribution, possibly depending on \vec{t}
4. From a multinomial distribution with parameter \vec{z} , choose each of N words $w_i \in \{1, 2, \dots, K\}$. We will refer to this set of words as $\vec{w} \in \{1, 2, \dots, K\}^N$ as well.

This process is drawn as a plate diagram in Fig 5.1.

To evaluate the conditional probability of an image being associated with a tag, given its set of words, we calculate:

$$\mathbb{P}(t_i|\vec{w}) = \frac{\mathbb{P}(t_i)\mathbb{P}(\vec{w}|t_i)\mathbb{P}(\vec{w}|t_i)}{\mathbb{P}(\vec{w})} = \frac{\mathbb{P}(t_i)\mathbb{P}(N) \prod_{j=1}^N \mathbb{P}(w_j|t_i)}{\sum_{\vec{s} \in \{0,1\}^L} \mathbb{P}(\vec{s})\mathbb{P}(N|\vec{s}) \prod_{j=1}^N \mathbb{P}(w_j|\vec{s})} \quad (5.1)$$

We simplify this expression by assuming that the distribution of the N , the number of visual words assigned to the image, is independent of \vec{t} , allowing us to remove the probabilities involving N . We further assume that the denominator can be approximated

by a sum of probabilities over the t_k , those tag vectors with a 1 in position k . This is not a particularly accurate assumption, but as our current application demands only a relative confidence in each tag, this inaccuracy is non-critical and allows a software implementation to quickly calculate an approximate normalization. Thus, we use the expression

$$\mathbb{P}(t_i|\vec{w}) \approx \frac{\mathbb{P}(t_i) \prod_{j=1}^N \mathbb{P}(w_j|t_i)}{\sum_{k=1}^L \mathbb{P}(t_k) \prod_{j=1}^N \mathbb{P}(w_j|t_k)}$$

as an estimate for our confidence for assigning tag i to the image described by \vec{w} .

5.2 Implementation and Results

To calculate confidences for applying tags to a set of visual words \vec{w} , we estimate $\mathbb{P}(t_i)$ as the proportion of test images with the tag. Since the probabilities depend on such large products, they are sensitive to zero values in the probability estimates and in general depend rather strongly on the training data. Thus, we strongly weight our estimates for these probabilities toward the "no knowledge" distribution. This can also be thought of as smoothing the estimated distribution, a common practice in machine learning. We have no order for the words, so weighting towards the average is the only smoothing easily available. We take

$$\mathbb{P}(w_i|t_j) = \frac{1 - \alpha}{K} + \alpha W_{w_i,j} \quad (5.2)$$

Where $W_{w_i,j}$ is the estimate of the probability of a word having value w_i , given an image with tag i . It is produced by a simple counting procedure as in the 'ad hoc' algorithm.

The implementation requires the products to be weighted so that the products can be computed numerically. Thus, an equation closer to that implemented (in MATLAB) is

$$\mathbb{P}(t_i|\vec{w}) = \frac{\mathbb{P}(t_i) \prod_{j=1}^N \frac{1-\alpha}{K} + \alpha W_{w_j,i}}{\sum_{k=1}^L \mathbb{P}(t_k) \prod_{j=1}^N \frac{1-\alpha}{K} + \alpha W_{w_j,k}} \quad (5.3)$$

By experimenting with our training set (2686 images with one or more of a set of 10 tags, analyzed with SIFT to produce 2,144,991 total features clustered into 1000 visual words), we found an approximately optimal value α as 0.01. Results were comparable, but in most cases superior, to the ad hoc approach (see Fig 5.2)

Interestingly, we can see the ad hoc approach as an approximation to this new approach, where α is been chosen to maximize the relative importance of the linear combination of measured word probabilities. This is a reasonable approach when the data set is too small to capture the increasingly sparse interactions between tags and words. If we expand the product

$$\prod_{j=1}^N \frac{1 - \alpha}{K} + \alpha W_{w_j,i} = \left(\frac{1 - \alpha}{K}\right)^N + \alpha \left(\frac{1 - \alpha}{K}\right)^{N-1} \sum_{j=1}^N W_{w_j,i} + \dots \quad (5.4)$$

We find that the second term has a maximum coefficient when $\alpha = \frac{2}{N+1}$. Our estimate of 0.01 corresponds to $N = 201$, which is comparable to the mean number of words (798) in images from our training set.

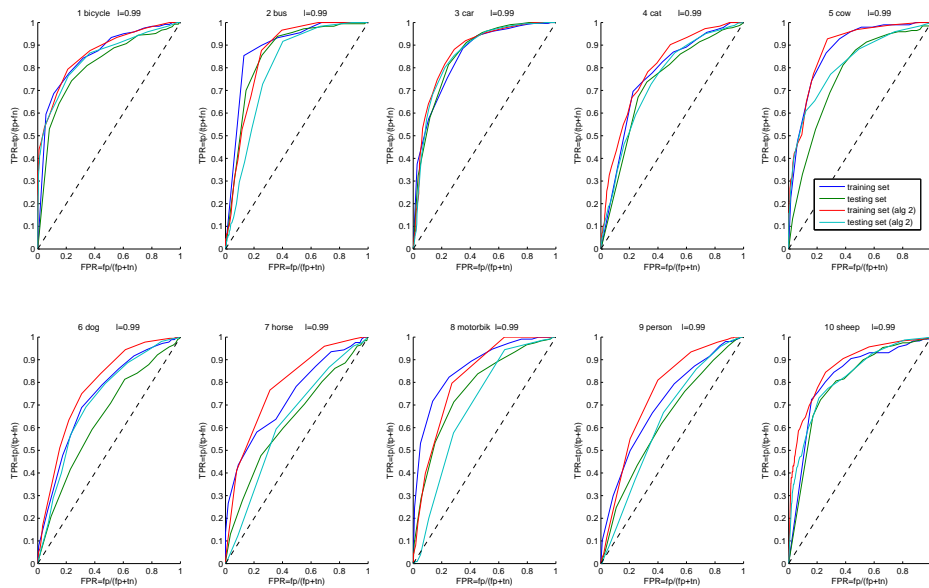


Figure 21: ROC curves for the tag prediction tasks, tested using two different algorithms on the training data and on a separate set of test data

6 Latent Dirichlet Allocation

Latent Dirichlet Allocation attempts to estimate the parameters of a more mathematically justifiable model, based on the assumption of a set of governing 'topics' allocated according to a Dirichlet distribution. We will not describe this method in detail here, but see [1], where it was introduced.

6.1 The Algorithm

LDA assumes that documents are generated by this process, visualized as a plate diagram in Fig 6.1:

1. Choose $N \sim \text{Poisson}(\xi)$ (Number of words in a document, not critical)
2. Choose $\theta \sim \text{Dir}(\alpha)$ (this gives the influence of each topic)
3. for each word w_n , $n = 1, \dots, N$,
 - (a) Choose a topic $z_n \sim \text{Multinomial}(\theta)$
 - (b) Choose a word with $p(w_n|z_n, \beta)$, a multinomial probability conditioned on the topic

The conditional probabilities required for prediction are intractable and so require approximate methods to estimate. Fortunately, many methods are available.

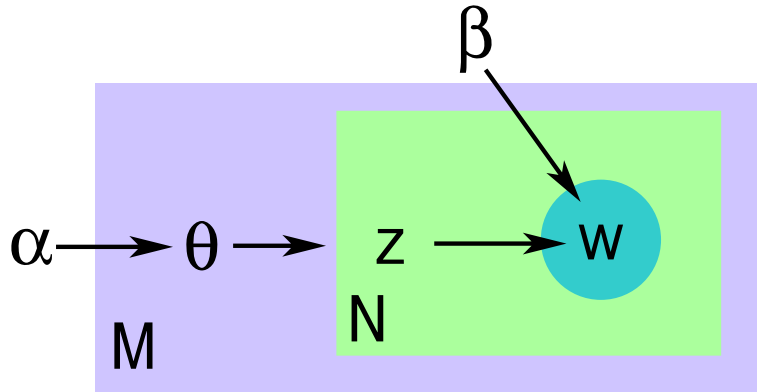


Figure 22: Plate Diagram for Latent Dirichlet Allocation Generative Model

6.2 Implementation

As a test, a MATLAB LDA package was employed to compute topics for our training set. An analysis was performed to determine 1000 topics for our training set. Informal examination of the topics most associated with each shows promise, as data unassociated with any of the tags can now be associated with topics not strongly correlated with any tag. See Figs 23, 6.2, 25.

7 Conclusion and Future Work

An Image Annotation Engine was described and implemented, using both an ad hoc algorithm which nonetheless produced reasonable results, and a slightly improved algorithm based on a probabilistic model. Evaluation of the engine was performed using the VOC2007 training data, using only global tagging information. This global approach has the advantage of requiring a minimal amount of metadata and as such lends it to application to any available set of tagged images. However, this global approach suffers from an excess of visual information unrelated to the supplied tags and as such requires at least a larger training set. Future work would seek to introduce local information in images and introduce a true topic model with latent topics, only a subset of which correlate highly with the set of supplied tags. Additionally, it is also in our interest to develop algorithms that do not require clustering and that can overcome the noisy data.

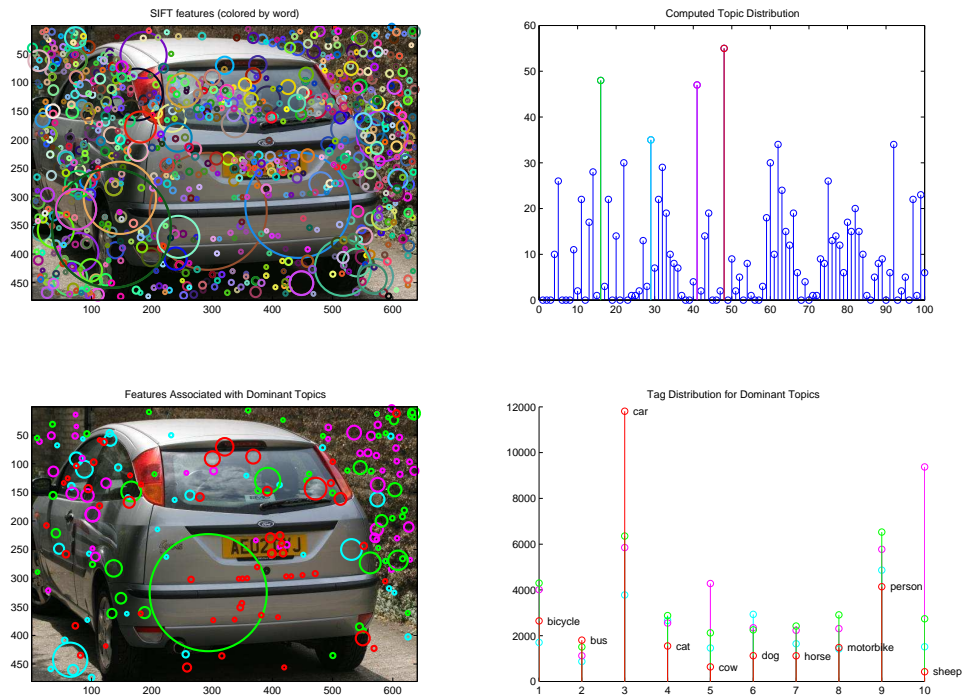


Figure 23: Example of LDA topic analysis

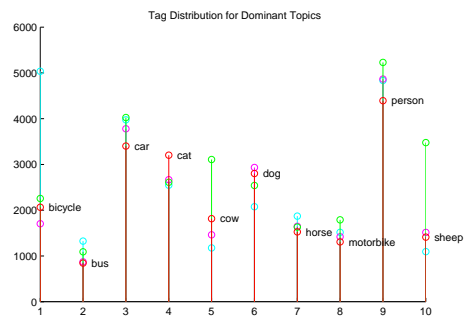
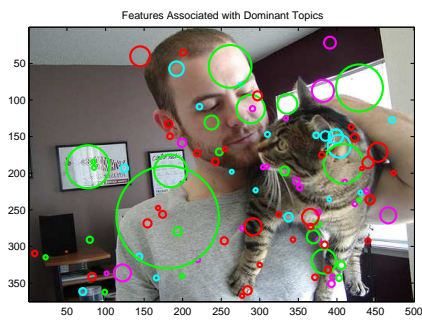
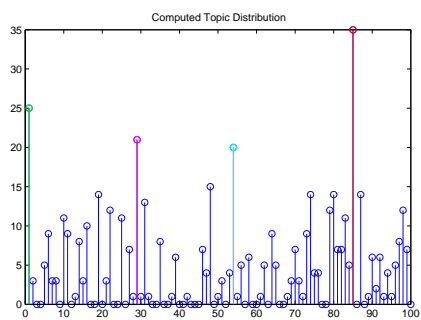
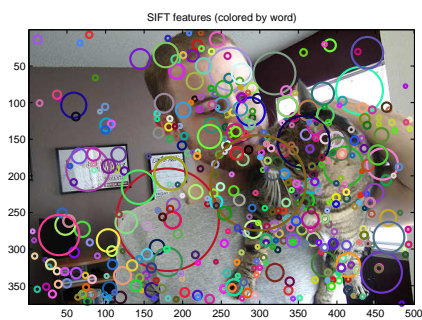


Figure 24: Example of LDA topic analysis

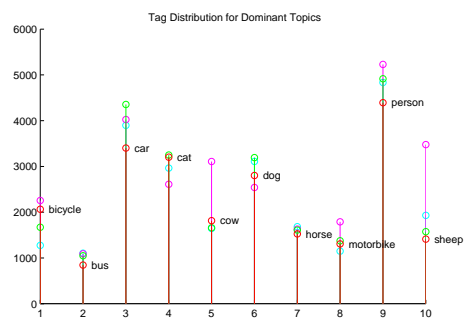
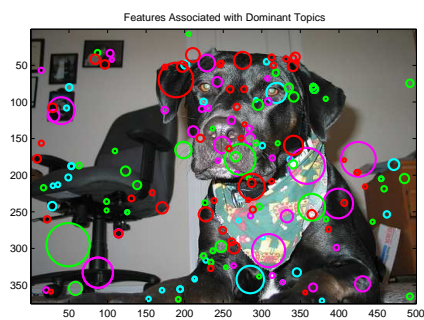
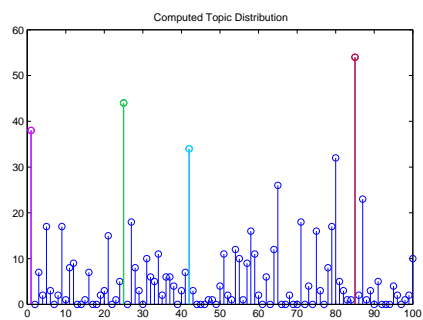
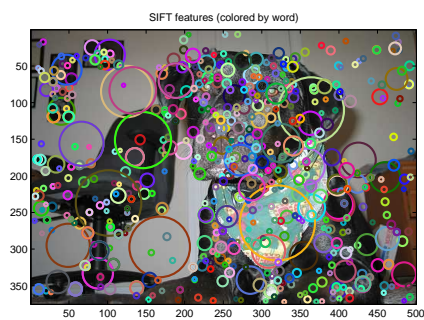


Figure 25: Example of LDA topic analysis

References

- [1] David M. Blei, Andrew Y. Ng, Michael I. Jordan, and John Lafferty. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:2003, 2003.
- [2] M. Everingham, A. Zisserman, C. K. I. Williams, and L. Van Gool. The PASCAL Visual Object Classes Challenge 2006 (VOC2006) Results. <http://www.pascal-network.org/challenges/VOC/voc2006/results.pdf>.
- [3] Michael Greenspan and Mike Yurick. Approximate k-d tree search for efficient icp. *3D Digital Imaging and Modeling, International Conference on*, 0:442, 2003.
- [4] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.
- [5] Marius Muja and David G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *VISSAPP (1)*, pages 331–340, 2009.
- [6] Wikipedia. Scale-invariant feature transform — wikipedia, the free encyclopedia, 2009. [Online; accessed 29-July-2009].