Image Classification Project Report

Alex Karl Voskuil
Jingbang Liu
Carnegie Mellon University, 10-601 Machine Learning
avoskuil@andrew.cmu.edu,jingbang@cmu.edu

Abstract

In this project report we outline the methods, experiments, and final results of applying various classification, training, and extraction algorithms to solve an image classification problem. In so doing we developed and compared four different classifiers using Gaussian Naive Bayes, K-Nearest Neighbors, SVM, and Logistic Regression classification techniques respectively, the former three of which achieved higher than baseline accuracy. We experimented with various ways to account for low testing accuracy and combat overfitting using methods like boosting, L2 regularization, and various feature extraction techniques such as HOG and bag of words. Some of these methods when applied to our classification algorithms proved useful while others did not. Ultimately we found our implementation of Gaussian Naive Bayes with boosting worked best for the image classification task.

Introduction

Throughout this semester we developed an understanding of many useful algorithms for classification and the application these algorithms might have. One such application is in the classification of images. In this project we seek to classify a large corpus of images, using the CIFAR-10 dataset containing 10,000 different images. We sought to classify these images into 1 of 10 correct classes (including, airplane, dog, etc.) using various machine learning techniques developed over the course of the semester.

In developing and testing various algorithms to solve this problem, we looked into other attempts in academia to achieve high image classification accuracy. One of the earlier ideas, outlined in our project proposal and taken from Gabriella Csurka's 2009 paper on feature extraction, was using Scale invariant feature transform (SIFT) due to its robustness to scale, rotation, illumination, and viewpoint invariances, to extract descriptors, using k-means clustering to find centroids in said descriptors and creating a bag of words to use in our classifier from these centroids as our features (Csurka). In this report we used this method but also chose to use the easier to implement Histogram of oriented gradients (HOG) feature extraction method as well. We also took inspiration from Le Hoang Thai's paper on using Support Vector Machines (SVM), which achieved nearly 86% accuracy, Elma Ferously's paper on implementing K-Nearest Neighbors with Naive Bayes to achieve similar accuracy, and Joseph Perla's paper on adaptive gradient descent. We also looked into our earlier work in this class, principally simple classification algorithms such as Gaussian Naive Bayes and Logistic Regression, to achieve satisfactory results. Ultimately for our submission, we used the K-nearest Neighbors, SVM, and a combined version of Gaussian Naive Bayes/K-Nearest Neighbors approaches for classification utilizing the HOG feature extraction method. We also

developed an interesting implementation of Logistic Regression which, while underperforming with 27% accuracy, utilized our bag of words approach and proved to be important to our understanding of the image classification problem.

Using our three other algorithms we have achieved better-than-baseline, or near-baseline-accuracy. In the rest of the paper we outline the four methods of classification and feature extraction. We then step through the results of our algorithms and the experiments we did to improve accuracy and combat overfitting.

Methods

In starting with this project we began by developing simple algorithms used earlier in the semester such as Naive Bayes and Logistic Regression, and progressed into more complicated classifiers such as K-Nearest Neighbors to achieve high classification accuracy. We then re-evaluated these simpler classification algorithms and applied boosting to help increase performance. In doing so and experimenting with different feature extraction methods we found that certain algorithms performed better than others and some were more susceptible to overfitting. We begin by looking at these simple algorithms and compare which performed best.

Model 1: Gaussian Naive Bayes Classifier with K-Nearest Neighbors Boosting

First, we began by using a Gaussian Naive Bayes classifier (classify1.m and train1.m in our submission) with features extracted from our database of images using VL_feat's built in HOG feature-extraction tool, vl_hog. After extracting said features in the form of a 1 x 496 vector per image, we trained our classifier on this data, which first calculated the estimated priors of the classes, Y, where the priors are represented below (1a).

$$\pi_k = P(Y = y_k) \tag{1a}$$

We then calculated the maximum likelihood estimates (MLE) of the mean and variance of each gaussian representing our HOG feature, X, using a helper function likelihood, where given,

$$\mu_{ik} = E[X_i | Y = Y_k] \tag{2a}$$

$$\sigma_{ik}^2 = E[(X_i - \mu_{ik})^2 | Y = y_k] \tag{3a}$$

Our maximum likelihood estimates are,

$$\bar{\mu}_{ik} = \frac{1}{\sum_j P(Y^j = y_k)} \sum_j X_i^j P(Y^j = y_k) \tag{4a}$$

$$\bar{\sigma}_{ik}^2 = \frac{1}{\sum_j P(Y^j = y_k)} \sum_j (X_i^j - \bar{\mu}_{ik})^2 P(Y^j = y_k) \tag{5a}$$

After storing our maximum likelihood estimates for the means and variances in a matrix, and the priors in a vector we store these values into our model structure thus completing training. For classification we then use this model structure and a newly extracted testing dataset of images, X, and apply the standard Naive Bayes Algorithm (6a).

$$\bar{Y_i} \leftarrow argmax_{y_k} P(Y = y_k) \prod_i P(X_i | y_k) \tag{6a}$$

Where the probability of a feature given a certain image class may be calculated as,

$$P(X_i|y_k) = N(X_i; \mu_{ik}, \sigma_{ik}^2) \tag{7a}$$

This method achieved baseline accuracy, but we sought to improve these methods using boosting. To do this we took inspiration from the Thai paper and combined Naive Bayes with K-Nearest Neighbors. This was done by extracting the features in the way mentioned above, but instead of using this raw data as the training data we added an additional step shown below in Figure 1, where the HOG features are the initial training dataset. In this additional step we use the numerical attributes of the HOG training data to obtain the k-nearest neighbor of a given image. We then do this for every image in our training data and then obtain the set of K observations as training data, and use it to build a model exploiting the Naïve Bayes algorithm based only on these categorical attributes (Figure 1).

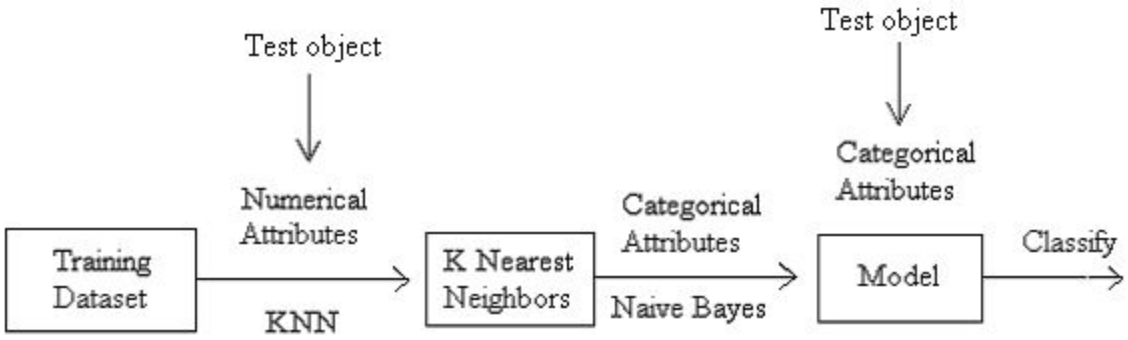


Figure 1. Graphical representation of the KNN+Gaussian Naive Bayes algorithm

While a simple modification, this method had powerful implications and greatly increased the accuracy of our classification algorithm.

Model 2: Logistic Regression with Bag of Words

The second method we used for our project, was a logistic regression classifier (classifyLR.m and trainLR.m). The method for feature extraction proved a bit more complex where we instead used VL_feat's built-in vl_phow feature extraction tool which extracted phow features (a dense SIFT feature applied at multiple resolutions) as descriptors. We then ran VL_feat's built-in kmeans clustering algorithm to cluster our descriptors into 186 different clusters by first finding the cluster means,

$$\vec{\mu} = \frac{1}{C_k} \sum_{i \varepsilon C_k} \vec{x}_i \tag{1b}$$

We then reassign the descriptors to the clusters using the objective function,

$$Obj = \sum_k \sum_i ||x_i - u_k||_2^2 \tag{2b}$$

After finding our clusters we then use each centroid as a "dictionary" for our bag of words. The final step of feature extraction involves creating a histogram feature vector. To create this vector we go through all 128 descriptors for a particular image and for every one of the 186 features of a descriptor, we find the closest cluster using vl_alldist, which calculates the euclidean distance between said feature and every centroid. We then add a count to the respective index of the histogram feature vector that minimizes the euclidean distance.

$$d(x, y) = \sqrt{\sum_{i=1}^{p} |x_i - y_i|^2} \tag{3b}$$

Finally, after extracting the features of our image in the form of a multinomial histogram feature matrix, X, we then apply our logistic regression classifier. We begin training our data by finding the weight matrix which we will need in the classification step to find the conditional probability of a class. We do this by finding the maximum conditional likelihood estimator (MCLE) of the weight matrix using gradient ascent,

$$w_{ik} \leftarrow w_{ik} + \eta \sum_{l} X_i^l (Y^l - P(Y^l = y_k | X^l, W)) \tag{4b}$$

To calculate P(Y = y_k | X), we apply the same logic to calculate the binary class case,

$$P(Y = 1|X) = \frac{1}{1 + exp(w_0 + \sum_{i=1}^{n} w_i X_i)} \tag{5b}$$

$$P(Y = 0|X) = \frac{exp(w_0 + \sum_{i=1}^{n} w_i X_i)}{1 + exp(w_0 + \sum_{i=1}^{n} w_i X_i)} \tag{6b}$$

With the exception that, because we have multiple classes not just two, we incrementally find the conditional probability that the class is y_k and that the class is the zeroth class where y_k is set to 1 and all other classes in Y are set to zero. After finding the weight matrix we store it and the dictionary into the model structure.

For classification then, we simply extract an image in much the same way as before, except that we do not generate a new dictionary but simply use this bag of words to create our histogram matrix as our feature matrix, X. For classification then, we simply take this X and our weight matrix and using the above described sigmoid method for finding the conditional probability of a class y_k given certain features, for Y we return,

$$Y_i \leftarrow argmax_{y_k} P(Y_i = y_k | X_i) \tag{7b}$$

Like the previously mentioned Gaussian Naive Bayes algorithm, we tried to improve accuracy by adding an additional step where we used K-Nearest Neighbors for boosting (classify2_LRKNN.m and train2_LRKNN.m) While this ultimately did significantly improve performance, as we will show later this had the unfortunate side effect of incredibly long running time.

Model 3: K-Nearest Neighbors

Prior to the boosting methods mentioned above, we began experimenting with different algorithms for classification. In this case, we decided to use the K-nearest neighbor approach (classify.m and train.m). We began working with this model by extracting the image features using the same VL_feat built in HOG feature-extraction tool, vl_hog, as we used in the Gaussian Naive Bayes Classifier.

After feature extraction of the training and testing data, we use this raw data to go through every test image's data and find the k-nearest image neighbors in the training set. We do this by calculating the Euclidean distance for the image against the entire training set, and then sorting the class of each training image by its distance from the testing image as shown below (3a).

$$NN_{test} = sort_{ascending}(\sqrt{\sum_{j}^{f} |x_{test,j} - x_{train,j}|^2} | \forall y_{train} \in TrainSet) \quad (1c)$$

Then using a given K selected via cross-validation we add the K-nearest neighbors into a set with their respective classes. Finally instead of using the means approach, we choose a class via a histogram "voting" method where the class with the highest observations is chosen as the testing image's respective class (2-3c).

$$KNN_{test} = NN_{test}[1:K] \quad (2c)$$

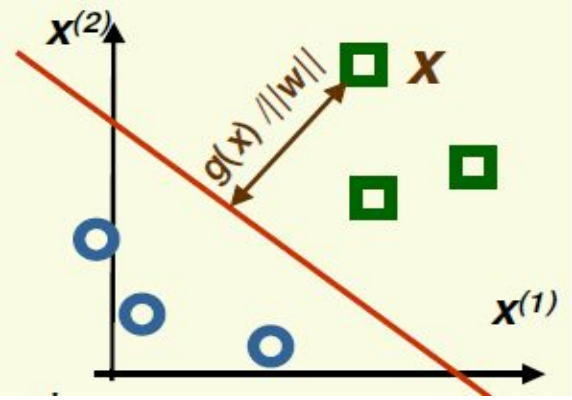
$$y_{test} = mode(y_i | \forall i \in KNN_{test}) \quad (3c)$$

This model proved to be an incredibly simple but powerful method that we later used to combine with the former two models in the form of boosting to increase accuracy.

Model 4: Support Vector Machine

In our final model, due to the widespread interest in SVM across the papers we have read, we decided to implement our own (classify2.m and train2.m). We begin by extracting the features using the the raw vl_hog, HOG extraction method as mentioned above. From here we wish to create a hyperplane to linearly separate our data. To make sure our results are generalizable we wish to maximize the margin of this hyperplane, and find the support vectors, or images closest to the hyperplane as shown below (Figure 2).

Figure 2: Graphical Representation of the hyperplane and maximizing margins problem, including the equation for the classifier (1d).



$$g(x) = w^t \cdot x_{test} + b \tag{1d}$$

Using this hyperplane we may then determine if a given image is a certain class depending upon what side of the plane it lies, or more formally;

$$y_{test,class} = \begin{cases} true | g(x) \geq 1 \\ false | g(x) \leq -1 \end{cases} \tag{2d}$$

However, because we wish to maximize the margins of our plane the problem becomes an optimization problem in the form below;

$$min_w(\frac{||w||^2}{2}) \tag{3d}$$

Constrained by, $y_i(w^t \cdot x_i + b) \geq 1$

Now to find the optimal w value, we use the Kuhn-Tucker theorem to convert *w* into the form;

$$w = \sum_i^n \alpha_i y_i x_i \tag{4d}$$

where we seek to maximize alpha and by extension the margins using this formula.

$$max_{alpha}(\sum_i^n \alpha_i + \frac{1}{2}\sum_i^n \sum_j^n \alpha_i \alpha_j x_i x_j y_i y_j) \tag{5d}$$

Constrained by, $\begin{Bmatrix} \alpha_i \geq 0 \\ \sum \alpha_i y_i \geq 0 \end{Bmatrix}$

Now that we have the problem explained theoretically, we can now solve the problem algorithmically. To do this, we will create variables H and f to represent the optimization problem, and A,a,b and B to represent the constraints. Doing so the problem above now becomes;

$$min_{alpha}(\frac{1}{2}\alpha^t H \alpha + f^t \alpha) \tag{5d}$$

Constrained by, $\begin{cases} A\alpha \leq a \\ B\alpha \leq b \end{cases}$

In this form then we may call the quadratic programming function, quadprog to solve for *alpha*. With *alpha* solved for, we now may solve for w as shown above. With these solved, we finally use the discrimination function mentioned above to choose a class for our respective image.

The only difference from this implementation is that, like the logistic regression method, both classifiers are commonly used for binary data. Just like the Logistic Regression classifier however this is easily accounted for by simply turning the problem into a “one vs all” problem where we loop through each class and in the end choosing

the class that has the highest g(x) value amongst all classes. While initially this model gave us low testing accuracy, we will describe how we used boosting to achieve higher than baseline accuracy.

Experiments and Results

We examined our results by combining the provided 5,000 images in the small_batch Matlab files as training data and using CIFAR-10 dataset as testing data for our four respective algorithms.

For the Naive Bayes Classifier when comparing the training labels and the training classification results we obtained around 55-60% training accuracy. When comparing the testing labels and the testing classification results obtained approximately 48% testing accuracy. This achieved our designated baseline surprisingly well, but we decided to use KNN boosting as mentioned previously to increase our accuracy. This had the fortunate consequence of increasing testing accuracy by nearly 10% to 58.1% accuracy, with little effect on run-time. We decided to further experiment and see how adjusting the size of K affected our accuracy. Ultimately, we found that K=100 achieved optimal testing accuracy with accuracy decreasing with higher and lower K as shown below.

Value of K for K-Nearest Neighbors	k=25	k=50	k=100	k=1000	k=10000
GNB Testing Accuracy	48%	53%	58.1%	52%	45%

Table 1: Relationship between K and Testing Accuracy for GNB

As mentioned above the logistic regression method proved problematic, initially yielding very poor accuracy, around 10% training and testing accuracy. These issues were resolved by switching feature extraction methods from those used in the naive bayes algorithm to the one described above. Ultimately, we had to experiment with two main ways to increase accuracy; changing the step size for gradient ascent, and changing the K for the K-means clustering. First we experimented with using a naive implementation of adaptive gradient descent, where the step size decreased each iteration, instead of a constant step-size (*Perla*).

Next, we experiment with changing the size of K and found that large K, around 186, both increased accuracy and to our disadvantage, run-time. With a high K and comparing the training labels and the training classification results we ended up with approximately 60-80% training accuracy. However, this came at the cost of low testing accuracy around 20%. As one would imagine with high K values, this resulted in massive overfitting.

To combat this overfitting, besides lowering the size of K, we may also try to perform L2 regularization on the gradient ascent algorithm using the formula below in place of our current means of generating weights (1e).

$$w_i \leftarrow w_i - \eta\lambda w_i + \eta \sum_l X_i^l (Y^l - P(Y^l = y_k | X^l, W)) \tag{1e}$$

Where, lambda is the inverse of the variance of the matrix of the weights, W. This had the fortunate consequence of increasing accuracy to 27.1%, while not achieving baseline, the classifier provided a unique opportunity for us to apply many different tools from the course to solve it.

The K-nearest neighbor approach as mentioned above, proved very successful, initially achieving 44% testing accuracy when given the larger 5000 image training dataset, near-baseline. Like the boosting done in the Gaussian

Naive Bayes classifier, we also experimented with the size of K. We found that adjusting K to be equal to 5 achieved optimal accuracy and that increasing or decreasing K lead to less accurate classification. Due to our adjustments we were able to increase accuracy by nearly 10% to 53.1% testing accuracy.

Value of K for K-Nearest Neighbors	k=3	k=4	k=5	k=15	k=30
KNN Testing Accuracy	49%	51%	53.1%	52%	50%

Table 2: Relationship between K and K-Nearest Neighbors Testing Accuracy

Finally, in our last classifier while we initially and surprisingly achieved our lowest testing accuracy, we managed to improve accuracy to 52.5% accuracy. Our model showed dismal results with around 21% accuracy originally, we decided to boost our model with k-nearest neighbors implemented in a similar way to how we did the GNB model and this managed to increase our testing accuracy significantly. Ultimately our model achieved its highest accuracy between 50-75 values for K, with the optimal K-value we chose being set to K=65. We also tried working with the bag of words extraction method used in our logistic regression implementation, but found that performance actually worsened with around 10% accuracy.

Table 3: Final Results

Project Classifiers	Gaussian Naive Bayes + KNN Boosting	Logistic Regression + Bag of Words	K-Nearest Neighbors	Support Vector Machine
Testing Accuracy	58.1%	27.3%	52.9%	53.4%

Conclusions

Using our four very different algorithms we have reached a few surprising conclusions. First, that a gaussian naive bayes classification algorithm with HOG feature extraction is sufficient to achieve adequate baseline accuracy for our image classification problem, and that by adding K-nearest neighbors for boosting, we may achieve very high testing accuracy around 58% Second, we have found that a logistic regression algorithm has the potential to be useful in solving this problem, but currently has a massive overfitting problem that even when regularization is applied and the K in its K-means clustering extraction is adjusted, limits its usefulness. The feature extraction method we are using in conjunction with the logistic regression classifier however is very powerful and holds promise for a more accurate classifier. Finally we were able to see how useful boosting with K-Nearest Neighbors was for a variety of algorithms, where in the end all of our algorithms had significantly improved accuracy after utilizing this method. Ultimately, we found that when experimenting with Logistic Regression, GNB, SVM, and KNN, that oddly enough the easiest solution may also be the best, when applied with boosting and other means of increasing accuracy we have developed over the course of this semester.

References

Ferdousy, Elma Zannatul, Mafijul Islam, MD., and M. Abdul Matin. "Combination of Naïve Bayes Classifier and K-Nearest Neighbor (cNK) in the Classification Based Predictive Models." *International Journal of Advanced Computer Science and Applications IJACSA* 4.11 (2013): n. pag. Web. <http://ccsenet.org/journal/index.php/cis/article/view/25588>

Perla, Joseph, and 2014. "Notes on AdaGrad." (n.d.): n. pag. Web. <http://seed.ucsd.edu/mediawiki/images/6/6a/Adagrad.pdf>.

Okumura, Sho, Naoya Maeda, Kiyoshi Nakata, Kazunori Saito, Yohei Fukumizu, and Hironori Yamauchi. "Visual Categorization Method with a Bag of PCA Packed Keypoints." *2011 4th International Congress on Image and Signal Processing* (2011): n. pag. Web. <http://www.cs.princeton.edu/courses/archive/fall09/cos429/papers/csurka-eccv-04.pdf>.

Thai, Le Hoang. "Image Classification Using Support Vector Machine and Artificial Neural Network." (2012): n. pag. Web. <http://www.mecs-press.org/ijitcs/ijitcs-v4-n5/IJITCS-V4-N5-5.pdf>.