10601B Midway Report

Chao-Ming Yen
Department of Computational Biology
Carnegie Mellon University
Pittsburgh, PA 15213
cyen1@andrew.cmu.edu

Tongkai Shao
Institute of Software Research
Carnegie Mellon University
Pittsburgh, PA 15213
tshao@andrew.cmu.edu

Abstract

In this report, we trained three different classifiers: K-Nearest Neighbor (KNN), Logistic Regression (LR) and Neural Network (NN), with 5,000 image sample subset of CIFAR-10 dataset and analyzed the implementation method and performance of each classifier.

1 Introduction

1.1 Motivation

Based on what we have learned from class, considering the amount of data and our understanding on each classifier, we chose K-Nearest Neighbor, Logistic Regression and Neural Network as our first classifiers. All these classifiers are basic and easy to implement so that in the future, our current experience can help us try more advanced classifiers, such as boosting, support vector machine and deep learning.

1.2 Background and Related Work

The database CIFAR-10 is a famous sample template for image classification, which consists of 60000 32X32 color images in 10 classes, with 6000 images per class. Each individual image sample consists of RGB color model parameters and is vectorized to 3072 parameters with value from 0 to 255. From the original database, we randomly select 5000 images as our training sample.

2 Methods

2.1 Distance-Weighted K-Nearest Neighbors

K-Nearest Neighbors classifier is a very intuitive algorithm and easy to implement. In training phase, we save both features *xTrain* and labels *yTrain* of the entire training set as our model. In classification phase, we assign the label by looking the most frequent class of the top *k* closest neighbors. Sometimes the frequencies of different classes could be tied, and one way to resolve this issue is to use the summation of "voting power" in replace of direct counting. Here, we use the Euclidean distance $\|\mathbf{x_a} - \mathbf{x_b}\|$ as our distance measure.

Consider the *k* nearest neighbors of test data $\mathbf{x}$:

$$\{(\mathbf{x_1}, y_1), (\mathbf{x_2}, y_2), \dots, (\mathbf{x_k}, y_k)\}$$

We define the "voting power" of each class *j* on test data $\mathbf{x}$ as the summation of the distance inverse of *k* nearest neighbors that have label *j*:

$$P(\mathbf{x}, j) = \sum_{i=1}^{k} \frac{\delta(y_i = j)}{\|\mathbf{x} - \mathbf{x_i}\|}$$

The class that possesses the highest voting power is chosen to be the label of the test data.

2.2 Multinomial Logistic Regression

In multinomial Logistic Regression [1], we want to classify the data into k classes with discrete label $c \in \{0,1 \dots k-1\}$. For the case of CIFAR-10 k equals 10, and the likelihood of the particular class given specific weight is determined by:

$$p(c|\mathbf{x}, \mathbf{w_c}) = \begin{cases} \dfrac{\exp(\mathbf{x} \cdot \mathbf{w_c})}{1 + \sum_{c<k-1} \exp(\mathbf{x} \cdot \mathbf{w_c})} & if\ c < k-1 \\ \dfrac{1}{1 + \sum_{c<k-1} \exp(\mathbf{x} \cdot \mathbf{w_c})} & if\ c = k-1 \end{cases}$$

In our implementation, we prepend the training and test data matrix with an "ones" column so that we can simplify

The conditional log likelihood can be expressed as

$$l(\mathbf{w}) = \log \prod_i \mathrm{p}(D|\mathbf{x}, \mathbf{w})$$

To maximize the conditional log likelihood of the weight vector **w**, we take partial derivative on each dimension j and each class c:

$$\frac{\partial}{\partial w_{j,c}} \prod_{i=1}^{n} \mathrm{p}(D|\mathbf{x}, \mathbf{w}) = \sum_{i=1}^{n} \frac{\partial}{\partial w_{j,c}} \log \mathrm{p}\left(c^i \middle| \mathbf{x^i}, \mathbf{w}^{c^i}\right)$$

where

$$\begin{aligned} \frac{\partial}{\partial w_{j,c}} \log \mathrm{p}\left(c^i \middle| \mathbf{x^i}, \mathbf{w}^{c^i}\right) &= \frac{\partial}{\partial w_{j,c}} \log\left(\frac{\exp\left(\mathbf{x^i} \cdot \mathbf{w}^{c^i}\right)}{1 + \sum_{c'<k-1} \exp\left(\mathbf{x^i} \cdot \mathbf{w}^{c'}\right)}\right) \\ &= \frac{\partial}{\partial w_{j,c}} \log\left(\exp\left(\mathbf{x^i} \cdot \mathbf{w}^{c^i}\right)\right) - \frac{\partial}{\partial w_{j,c}} \log\left(1 + \sum_{c'<k-1} \exp\left(\mathbf{x^i} \cdot \mathbf{w}^{c'}\right)\right) \\ &= \frac{\partial}{\partial w_{j,c}} \left(\mathbf{x^i} \cdot \mathbf{w}^{c^i}\right) - \frac{1}{1 + \sum_{c'<k-1} \exp\left(\mathbf{x^i} \cdot \mathbf{w}^{c'}\right)} \frac{\partial}{\partial w_{c,j}} \left(1 + \sum_{c'<k-1} \exp\left(\mathbf{x^i} \cdot \mathbf{w}^{c'}\right)\right) \\ &= x_j^i \delta\left(c = c^i\right) - x_j^i \frac{\exp\left(\mathbf{x^i} \cdot \mathbf{w}^{c^i}\right)}{1 + \sum_{c'<k-1} \exp\left(\mathbf{x^i} \cdot \mathbf{w}^{c'}\right)} \\ &= x_j^i (\delta\left(c = c^i\right) - \mathrm{p}\left(c^i \middle| \mathbf{x^i}, \mathbf{w}^{c^i}\right)) \end{aligned}$$

By gradient ascent, we can approximate the optimal solution by iteration:

$$w_j \leftarrow w_j + \eta \sum_{i=1}^{n} x_j^i (\delta(c = c^i) - \mathrm{p}\left(c^i \middle| \mathbf{x^i}, \mathbf{w}^{c^i}\right))$$

2.3 Artificial Neural Network

Artificial Neural Network is popular classifier that is used to estimate the class of data, which depends on a large number of inputs and known output label to train the data sets. It is generally described as interconnected neurons which can exchange data and messages between each node. The connections of nodes from different layers have numeric weights value that can be tuned based on previous experience, making neural network capable of learning [2].

There are several equations that often used in neural network classifier.

Hidden layer input function:

$$X(j) = \sum i\, \omega(ij) * X(i)$$

Hidden layer output function (sigmoid unit function) and output layer function:

$$X(j) = \frac{1}{1 + \exp\left(\omega 0 + \sum i \omega(i) * X(i)\right)}$$

$$Y(i) = \sum i\, \omega(i) * X(i)$$

Gradient descent method is generally used to update node weight value. Here are the functions of this algorithm.

For each output unit m:

$$\delta(m) = o(m) * \big(1 - o(m)\big) * (t(m) - o(m))$$

For each hidden unit h:

$$\delta(h) = o(h) * \big(1 - o(h)\big) * \sum k\, \omega(h, k) * \delta(k)$$

For each network weight $\omega(ij)$:

$$\omega(ij) = \omega(ij) + \Delta\omega(ij)$$

where

$$\Delta\omega(ij) = \eta\delta(j)X(i)$$

And for those functions, X means input data, t(k) means target output data of training example k, o(h) means observed unit output of training example h, $\omega(ij)$ means the weight from i to j and η means learning rate in this neural network classifier.

3 Experiments and Results

Feature extraction. Before we train the classifier, all 5000 image samples are preprocessed using VLFeat toolbox to extract HOG (Histogram of Oriented Gradient) [3] with cell size equals 8 as new sample feature.

Cross validation. In order to verify the performance of our classifiers, we partitioned the 5000 image samples into 5 batches, and used the first 4 batches as training set and the last batch as test set.

Normalization. In both training set and test set, each HOG features are normalized column-wise to real number in [0, 1], where the maximum feature becomes 1 and the minimum feature becomes 0.

3.1 Distance-Weighted K-Nearest Neighbors

Here, we compared the native KNN and distance-weighted KNN by running grid search on $k = \{5, 6, \ldots 30\}$ using both KNN methods to classify the training set. The performance was evaluated with test set to determine which version of KNN combined with which *k* produces the best result.

3.2 Multinomial Logistic Regression

In addition to *batch gradient ascent* that we have discussed in section 2.2, we also tried *stochastic gradient ascent* in our logistic regression implementation. Instead of summing up the gradient shifts of whole training data, we update the gradient by looking at one sample at a time to speed up the computation time.

3.3 Artificial Neural Network

Label conversion. In order to extend the range of original data labels, which is a 4000*1 matrix, we converted it to a 4000*10 matrix. For each row, if the original label value is four, the fifth element of this row will be 1 and all others are 0. As a result, we got a 4000*10 matrix with all elements are 0 or 1 as new label set.

Neural network structure. The neural network has 3 layers, input layer, hidden layer and output layer. The input is a 4000*496 matrix, so the input nodes n should be 496, the hidden layer nodes h has n + 1 nodes, which is 497. Because we converted the labels data to a 4000*10 matrix, the output nodes m has 10 nodes.

Function and parameters. We used sigmoid function as driving function and gradient descent algorithm to update nodes and weights. The learning rate η equals to 0.1.

3.4 Results

To evaluate the accuracy of each classifier, we used trained classifiers to classify the test set and construct confusion matrix with classified labels and the test labels.

The result of grid search with native KNN and distance-weighted KNN is shown below (Figure 1), we found that the performance of distance-weighted KNN (mean = 0.5598) is slightly better than native KNN (mean = 0.5540). The best k is when $k = 23$, which gave the best performance in both version of KNN.

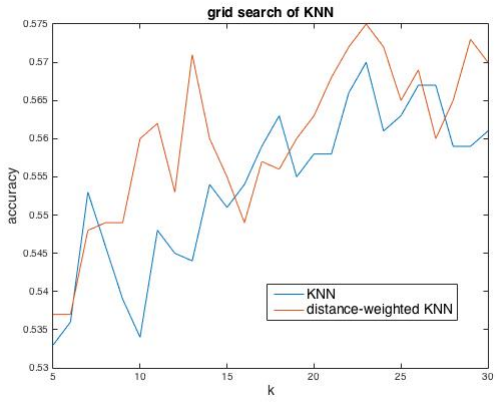

Figure 1: Grid search result on k of different KNN methods

The following table (Table 1) demonstrates the performance of three trained classifiers running on the test set. Currently our best performance is using distance-weighted KNN with k = 23, which has been submitted to Autolab and get score 52/100.

Table 1: Performance of three different classifiers

Classifier	Accuracy	Time	Remark
KNN	57.5%	1 ~ 2 min	Using distance-weighted KNN with k = 23
LR	12.8%	~ 40 min	Classification result are centralized in one or two classes
NN	13.3%	~ 13 min	Classification result are centralized in one or two classes

4 Conclusion

From the result, we can see that only K-Nearest Neighbor classifier passed the baseline criteria (48 %, using Naive Bayes classifier), the accuracy of other classifiers are much below the baseline. For LR and NN we found that the accuracy is around 10 % and it means both classifiers are no better than blind pick. Besides, most classified labels are centralized in one or two classes.

One possible reason why our last two classifiers failed is that the calculation result of sigmoid function, most equal to zero, cannot update the parameters effectively, thus the weight vector barely updates throughout the gradient descent/ascent process and the classifier tend to "stay" at the initial state. Currently we are looking for couple numerical solutions that can help us solve the problems of extremely biased sigmoid function output. Moreover, in the future we want to try implementing other classifiers such as SVM, boosting and Deep Learning, which are considered to be more accurate classifiers for image classification.

References

[1] Bob Carpenter. (2008), Lazy sparse stochastic gradient descent for regularized multinomial logistic regression. *Technical report, Alias-i*.

[2] A. Hyvarinen & E. Oja. (2000), Independent component analysis: algorithms and applications. *Neural networks*, **13**(4-5):411–430.

[3] Navneet Dalal & Bill Triggs (2005), Histograms of Oriented Gradients for Human Detection, Computer Vision and Pattern Recognition. *CVPR 2005. IEEE Computer Society Conference on (Volume:1)*