Single Object Recognition

Attinderpal Singh, Kasi Karuppiah Alagappan
Northeastern University
{singh.at, alagappan.k}@husky.neu.edu

Abstract

The objective of this paper is to build a Single Object Recognition Machine Learning algorithm and compare its results with similar implementations in scikit-learn and Google's TensorFlow. The algorithm is evaluated on MNIST and CIFAR-10 data set. The dataset is split into test and train subset. The algorithm trains on the training subset while the validation is done with the test subset. There are various ways to approach this problem. We will discuss few of them and which one we used for our classifier.

1 Introduction

Many industries face a problem of analysis or classification of binary data based on images. Humans are extremely capable in distinguishing objects. Transferring these capabilities efficiently has always been a challenging task. By transferring the similar recognition capability to the machines saves a lot of time and free workforce from tedious tasks.

1.1 Motivation

Binary image classification or single object recognition can be used in various fields like detecting a cancer cell, detecting fake currency, rejecting bad samples from a conveyor belt and so on.

1.2 Challenges and Results

Major challenges with object recognition is feature extraction from data set. But if a large enough data set is collected then this can be transformed to less challenging classification/regression problem. Which can be solved with various Machine Learning techniques. We were able to achieve comparable results with inbuilt scikit-learn regression model and Complex Convolutional Network(CNN) implementation in TensorFlow.

2 Related Work

Different approaches have been implemented for object recognition. All of them can be used for our use case but our approach is simpler in implementation and results are in the ball park of those implementations.

- Decision tree
- SIFT
- Logistic Regression
- CNN
- R-CNN

Decision trees were considered for the project. They are flowchart like structures in which an internal node represents a test on an attribute. Each branch represents outcome of a test and leaf nodes represent labels of each class. The path from node to root represent classification rule. They get good results if the features are less. For MNIST dataset there are nearly 750 features. Its efficiency gets really bad if we have features in the range of couple thousands.

SIFT (Scale Invariant Feature Transform) is very efficient if the dataset is small. SIFT key points of objects are first extracted from the images and stored in the database. Object is recognized by comparing each feature from new image with the database. Prediction is done based on the Euclidean distance of these feature vectors. From the whole set of matches, subset of key points that agree on object and its location, scaled and orientation in the new image are identified as good matches[6]. It has an advantage of scale and rotational invariance. The drawback of SIFT is that its computationally intensive and mathematically heavy. SIFT is based on histogram of gradients i.e. the gradient of each pixel in a zone has to be calculated. And moreover, it's a patented algorithm.

CNN and R-CNN (Regions with CNNs) are different way to approach this problem. Recent advances in CNN design, notably deeper models with more layers by the availability of cheap computing and enhanced techniques such as inception modules have created models that rival the accuracy of human object identification. Moreover, CNN are making a big change in the area of self-driving vehicles to medical imaging evaluation. In these areas computers are already out performing best human rivals. In object recognition Regions with CNNs is another approach which increases the accuracy of CNNs substantially. At a high-level R-CNN looks at the image through a window of different sizes and for each size it

tries to group together adjacent pixels by texture, color, or intensity to identify the objects. There are variants of R-CNN like Fast R-CNN, Faster R-CNN and Mask R-CNN [2].
We compared the results of CNN with our regression approach.

3 Approach

Logistic regression is a predictive analysis regression model used to estimate the probability of a response based on one or more independent features. In binary logistic regression the outcomes are labelled 0 or 1. If an outcome is favorable then it is coded as 1 and otherwise as 0.

3.1 Logistic Function

In logistic regression we use the following hypothesis to predict the probability that a given example belongs to the class 1 versus the probability that it belongs to class 0.

$$P(y=1|x) = h_\theta(x) = \frac{1}{1+\exp(-\theta^\top x)} \equiv \sigma(\theta^\top x),$$

$$P(y=0|x) = 1 - P(y=1|x) = 1 - h_\theta(x).$$

The above two equations could be condensed into a function often called sigmoid function.

$$\sigma(z) \equiv \frac{1}{1+\exp(-z)}$$

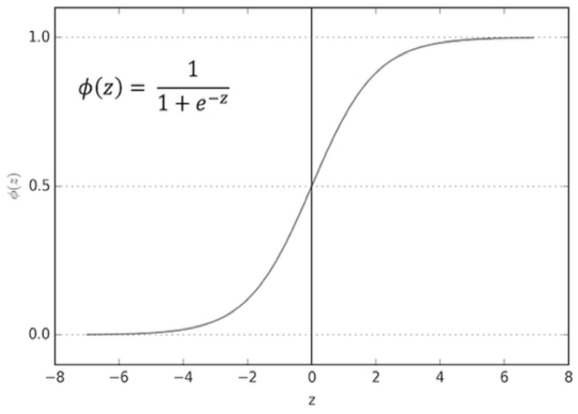

Fig 1
Sigmoid Function

It is an S-shaped function that fits the value of Tx into the range [0, 1] so we could interpret the h(x) as the probability.

3.2 Cost Function

Functions have weights and we need to find the best value that fits the data used for training, i.e. the weights that differentiate the probability of input x belonging to class 1 or class 0 with larger probability. Random values are picked at first and we measure how well the algorithm performs using those random weights. This measure is computed using the loss function

$$J(\theta) = -\sum_i \left(y^{(i)} \log(h_\theta(x^{(i)})) + (1-y^{(i)}) \log(1-h_\theta(x^{(i)}))\right)$$

For each input x one of the terms in the summation is non-zero. When y = 1, h(x) needs to be made large and when y = 0, h(x) needs to be made small.

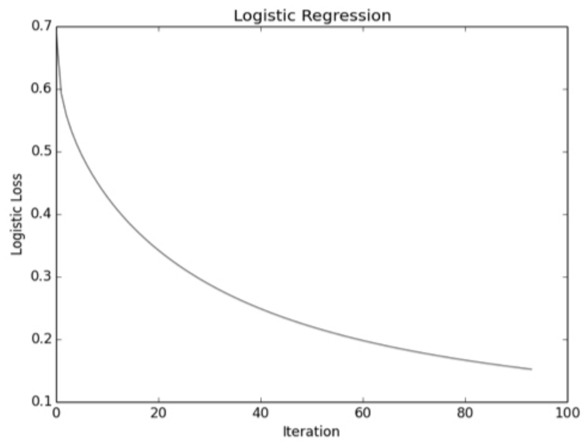

Fig 2
Logistic loss of MNIST dataset

3.3 Gradient Descent

The fastest way to minimize the cost function is by decreasing/ increasing the weights, i.e. fitting them better to the training data. The delta change in weights is given by the derivative of cost function with respect to each weight. It tells us how loss would change if the weights were modified based on the parameters.

$$\frac{\partial J(\theta)}{\partial \theta_j} = \sum_i x_j^{(i)} (h_\theta(x^{(i)}) - y^{(i)})$$

$$\nabla_\theta J(\theta) = \sum_i x^{(i)} (h_\theta(x^{(i)}) - y^{(i)})$$

3.4 Batch Gradient Descent

Batch gradient descent is a variation of the gradient descent algorithm that updates the weights after all training data have been evaluated instead of updating each training data. Using this approach reduces the number of updates made to the weights and results in a stable convergence. Moving to a batch gradient descent allows parallel processing when large amount of data needs to be processed.

3.5 Dataset

3.5.1 MNIST

The MNIST database is a set of 60000 training handwritten examples and a test set of 10000 examples. Each example is a 28x28 image with gray levels.
Data - A 784 column array storing gray level values in row major order.
Labels - A number in range of 0-9 representing the digit.

3.5.2 CIFAR-10

The CIFAR-10 dataset has 60000 32x32 color images in 10 mutually exclusive classes, with 6000 images per class. There are 50000 training images and 10000 test images.
Data - A 3072 column NumPy array storing the RGB channel values in row major order.
Labels - A number in the range of 0-9 representing the class the image belongs to.

3.6 Preprocessing

3.6.1 Normalization

The independent features were standardized to a common scale with an average of 0 and standard deviation of 1 using z-scores. The standard score is:

$$z = \frac{x - \mu}{\sigma}$$

where $\boldsymbol{\mu}$ is the mean and $\boldsymbol{\sigma}$ is the standard deviation. The mean and standard deviations of each feature from the training data was used to normalize the test data.

3.6.2 Constant Feature

A constant feature with value 1 was added to the dataset to help avoid separate calculations when adding bias value in the sigmoid calculation.

4 Evaluation, Testing and Results

4.1 Evaluation

4.1.1 K-Fold cross validation

Cross validation technique was used to assess the results of the logistic regression model. Cross validation involves partition of data subsets, performing analysis of one subset and validating the other subset. K-Fold cross validation is a variation of cross validation where the dataset is partitioned into k partitions and over each iteration a single partition is retained as validation data while the remaining k-1 partitions are used as training data.

Fold	Accuracy	Precision	Recall
1	97.30%	97.56%	97.30%
2	97.64%	97.89%	97.64%
3	97.61%	97.86%	97.61%
4	97.40%	97.72%	97.40%
5	97.49%	97.77%	97.49%
6	97.46%	97.71%	97.46%
7	97.33%	97.64%	97.33%
8	97.59%	97.85%	97.59%
9	97.19%	97.53%	97.19%
10	97.56%	97.78%	97.56%
Mean	97.46%	97.73%	97.46%
Standard Deviation	0.001431	0.001172	0.001431

Table 1
K-fold cross validation of MNIST in LR

The precision is the ratio tp / (tp + fp) where tp is the number of true positives and fp the number of false positives. The precision is intuitively the ability of the classifier not to label as positive a sample that is negative.
The recall is the ratio tp / (tp + fn) where tp is the number of true positives and fn the number of false negatives. The recall is intuitively the ability of the classifier to find all the positive samples. In this project accuracy, precision and recall values are calculated from confusion matrix.

4.1.2 Parameters

Learning rate - A constant value used in backpropagation to alter the speed of learning. If the value is too small the gradient descent can be slow and if it's too large the gradient descent may fail to converge.

Tolerance value - A constant value that is used to estimate
The algorithm is run for different learning rates and tolerance values to find the best combination of the two constants that increase the accuracy of the model avoiding overclocking the CPU.

Models	Learning Rate	Tolerance
MNIST	0.1	0.001
CIFAR-10	1.0	0.000001

Table 2
Parameters comparison for LR

4.2 Testing / Results

The accuracies of the LR implementation vs TensorFlow and LR implementation vs scikit-learn logistic regression of CIFAR-10 and MNSIT were compared and the results are in tables 3 and 4 respectively.

Models	Accuracy
TensorFlow	92.2%
LR Implementation	90.8%

Table 3
CIFAR-10 dataset comparison

Models	Accuracy	Precision	Recall
SKLearn	99.3%	99.3%	99.3%
LR Implementation	97.4%	97.7%	97.4%

Table 4
MNIST dataset comparison

So from above results we can see that our implementation did significantly well when compared to TensorFlow CNN and Sklearn LR implementations.

Acknowledgments

This project was implemented by Attinderpal Singh and Kasi Karuppiah Alagappan. The work was divided evenly. Attinderpal Singh worked on scikit-learn implementation of MNIST and LR implementation of CIFAR-10. Kasi Karuppiah Alagappan worked on LR implementation of MNIST and TensorFlow implementation of CIFAR-10. Preprocessing of datasets, documentation and implementation of LR in scikit-learn were done by both.

References

[1] Lowe, D. G. (1999). Object recognition from local scale-invariant features. In Computer vision, 1999. The proceedings of the seventh IEEE international conference on (Vol. 2, pp. 1150-1157). Ieee.

[2] Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. In Advances in neural information processing systems (pp. 91-99).

[3] Nasrabadi, N. M. (2007). Pattern recognition and machine learning. Journal of electronic imaging, 16(4), 049901.

[4] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.

[5] Krizhevsky, A., Nair, V., & Hinton, G. (2014). The CIFAR-10 dataset. *online: http://www.cs.toronto.edu/kriz/cifar.html.*

[6] Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, *60*(2), 91-110.

Appendices

MNIST dataset - loaded from scikit-learn library
CIFAR-10 - https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
LogisticRegression.py - logistic regression implementation with 10-Fold cross validation on MNIST dataset
SKLearnLogisticRegression.py - scikit-learn implementation of logistic regression on MNIST dataset
tensorplayground.py – TensorFlow implementation of CNN
LogisticRegressionCifar.py - logistic regression implementation on CIFAR-10 dataset
ModifiedLogisticRegressionCifar.py - logistic regression implementation on CIFAR-10 dataset working with normalized data to speed up the process