ECE 532: Term Project: Classifying Images in CIFAR-10 Dataset Using Supervised Algorithms

Final Report

Written by:
Nicholas Chelales

(a) N. Chelales

University of Wisconsin-Madison
December 1, 2020

Contents

1 Project Dataset

The project dataset to be used is the CIFAR-10 dataset (found here). The data is publicly available and consists of 10,000 32x32 RGB images, each belonging to mutually exclusive classes (1 of 10 possible options, see the dataset). The goal is to test the three different algoithms, K-Nearest Neighbor, Kernel Based SVM, and convolutional neural networks and measure their performance in classifying the images as one of the 10 possible classes.

2 Algorithm Investigation

2.1 K-Nearest Neighbors

The k-nearest neighbors algorithm works by making a prediction based off of the closest K training examples in the N-dimensional feature space (where N is the number of dimensions in the feature set). There are many different distance functions that are used in the k-nearest neighbors classification such as city block (Manhattan), euclidean distance, cosine distance, and mahanblois distance. A subset of these distance metrics will be selected and the performance will be compared among these metrics. In addition to distance metrics, the number of neighbors (K) will be a second parameter and performance will be compared among different number of neighbors to see the best performing K.

2.2 Kernel Based SVM

The kernel based SVM is a linear classifier which uses a kernel to convert a N-dimensional space into a higher dimensional space, making it easier to distinguish between different classes compared to lower dimensional spaces that can sometimes be more difficult to separate with a linear boundary. There are many different types of kernels, such as a linear kernel, polynomial kernel, and and radial basis kernel. These different kernels will then be compared and performance will be evaluated to find the most optimal parameter. In addition, optimization and regularization parameters (such as gamma) can be used as parameters that can be tuned and evaluated.

2.3 Convolutional Neural Network

A convolutional neural network is a neural network that uses a filter that is convolved (filtering) the input image in order to get a result that can then be classified as one of the possible classes. The neural network is trained and the filter kernel is tuned appropriately to get an end result kernel that is applied across each layer. There can be many layers in a neural network, so the number of layers will be one of the parameters that is explored to see the most optimal number of layers. Another parameter that will be explored is the loss function, which is used to help tune the filter and avoid overfitting. There are different loss functions that can be used, and several functions will be evaluated to determine which is the most effective.

2.4 Performance Evaluation

Performance will be evaluated using three different loss functions: 0/1 Loss, squared loss, and hinge loss. These loss functions will be used to get a representation of performance on training data (for baseline model performance), and will be used with k-fold cross validation on the testing data to get a measure of the robustness of each model/parameter experiment.

2.5 Features and Feature Selection

There are several possible features from an image that can be used as possible inputs for the classification algorithm. Primarily, the raw input pixel values contain significant information that can be used for inputs to the models (although there is room to add features by preprocessing some of the images with filters such as horizontal or veritcal line filters, extracting color schemes in the HSV or LAB colorspace, augmenting the

dataset, etc. if necessary to achieve better results). For K nearest neighbors, the raw pixel values will be used as the features, and each test image will be compared with the all training images using the specified distance function. For SVM, we can treat the input data as the matrix of values in the image, flattened into a 1D array (basically like reading a single vector of values, rather than a matrix). For the convolutional neural network, the features will also be the raw input pixel values, as the network will use these values as it builds the optimal filter.

2.6 Update 1 November 17th

subsectionWork Progress

I have finished implementing the k nearest neighborhood classifier. Some basic google searching indicated previous KNN classifiers for image sets do not particularly perform well (SVM and CNN perform better), considering that the KNN algorithm is a clustering algorithm and many different classes of images can have similiar pixel distributions that cannot be easily clustered (i.e. a yellow car versus a tan truck). Nonetheless, I was curious about whether the different color representations of an image could result in a better or even optimal classifier. Particularly, I focused on the hue,saturation, and value (HSV) color space, the light, green,blue (LAB), grayscale, and the standard RGB colorspace. The goal was to test whether any of these color spaces presented as possibly good features for clustering.In addition, I wanted to test the effect of the number of neighbors in KNN as well as the distance metric used (manhattan, euclidean, chebyshev and minkowski).

My process for KNN algorithm was as follows after data importing:

1. Convert the image to a colorspace representation (i.e. one of the 4 colorspaces mentioned above)
2. Get the histograms for each of the three represented color layers of the chosen color space
3. Vectorize all 3 histograms together to get a feature matrix, containing the concatenated histograms of each image in the rows
4. Normalized to standard (i.e. convert to 0 mean and unit variance)
5. PCA reduction to 2 principal components for dimension reduction and speed up computations
6. Perform a grid search to optimize the distance metric as well as the number of neighbors using 10 fold cross validation.
7. Save results of cross validation (see GitHub Page).
8. Test on the final holdout set (which contains 1000 images of each class), and generate a confusion matrix to gain a big picture result.

The results indicate about 20-30 percent accuracy for KNN, depending on the colorspace used. This is better than guessing, but is definitely not a great classifier. The idea behind using the histogram was two fold: using the histograms of each layer reduces the dimensions since each histogram has only 255 bins, instead of using the 32x32 = 3000 pixels for features. In addition, in the image processing sector, lots of information is used in histograms to transform one image to look like another, so it's not unreasonable to think some information could be used from the histogram in a classifier. On the flip side, possible reasons for low accuracy are the fact that color spectrums shift across many classes. If the classes were very distinguishable by color, this algorithm would likely perform better.

2.7 Potential Improvements to Current Work

Possible improvements to this algorithm could come from increasing the number of principal components. In this case, I neglected to look at the principal components to come up with a measurable quantity to justify choosing the right number of principal components to include in the feature vector. This opens up the potential for some improvement, although it is unlikely as KNN is not often used for direct image classification tasks.

2.8 Timeline

I am somewhat on track. It took me a while to get my feet underneath me again for an extended python project, so technically I am a bit delayed. But I have a good plan going forward and I am confident I can get the work done in time.

2.9 Update 2 December 1st

2.9.1 Improvements/Modifications to Prior Work

As mentioned in update number one, I did an SVD decomposition of the data to determine the most important principal components for each feature set. Prior to this, I arbitrarily chose to use 2 principal components, which was not really a good representation of the data and was an over-reduction of the data. After plotting magnitude of the singular values as a function of singular value index, I determined which principal components to keep by finding the value associated with the 'elbow' where the singular values significantly drop in magnitude, indicating unimportant components. Including more principal components increases the dimensionality of the data, and thus significantly increases the computation time, so I made the decision to reduce the cross fold validation from 10 to 3 for choose the best hyper parameters. This allows me to still have a strong method for choosing the best hyper parameters.

After making this modification, I was able to improve the correct classification rate to around 40 percent, which is a significant improvement from update number one's results. Updated confusion matrix and plots for the KNN classifier are available in the github drive.

2.9.2 New Work: SVM Methods

My second classification model is SVM. For SVM, I am testing several different hyper parameters, such as non-linear kernels (RBF, sigmoid, and polynomial), regularization parameter gamma influencing importance of data points to svm, and the margin regularizer C. I have several different values I would like to test for these parameters. For now, I only tested two different values of gamma and c, and tested the four different kernels using 3 fold cross validation to find the best parameters. I plan to expand the grid to include more values of gamma and c, but these results are not available yet due to high computational time (am working on a solution where I can train the model in the background while not diminishing other tasks on my machine). Plots and CV results are available in the github drive.

2.10 Timeline

I am on track to finish. Only one more classifier to implement and the report to write.

2.10.1 Week 1 Oct 26th

-Build project environment in PyCharm -Download and Import Data

2.10.2 Week 2 Nov 2

-Implement and test KNN algorithm

2.10.3 Week 3 Nov 9th

-Implement Convolutional Neural Network

2.10.4 Week 4 Nov 16th

-Implement Kernel Based SVM -Submit Update 1

2.10.5 Week 5 Nov 23rd

-Continue to implement Kernel Based SVM

2.10.6 Week 6 Nov 30th

–Implement Convolutional Neural Network -Buffer week for debugging/unexpected problems -Submit Update 2

2.10.7 Week 7 Dec 7th

-Write Report

2.10.8 Week 8 Dec 14th

-Submit Report

2.11 GitHub Page

Access here