Using Classification Algorithms to Identify Red Cars in Images*

Brianna Richardson

***Abstract*—With the influx of Big Data and the need to process and analyze it quickly and efficiently, Machine Learning is a growing domain. In particular, Image Analysis is an important pursuit with the plethora of video footage and images that can be utilized to retrieve critical information. In this paper, I tested a Supervised Classification algorithm to determine how accurate it was at predicting locations of red cars given training data with locations of red cars. With K-Nearest Neighbors (KNN) algorithm, I determined parameters required for a pixel to be considered part of a car, in terms of RGB value and neighboring pixels. Using the attributes from my training, I was able to successfully locate a small fraction of red cars in a subset of the testing image. While most all points found were actually pixels on a red car, this algorithm had a lot of false negatives, not identifying most of the red cars in the image. The results prove that KNN is an accurate, yet computationally expensive technique.**

***Index Terms*—Machine Learning, Supervised Learning, KNN, Obect Recognition.**

I. Introduction

IN the current Information Age that we find ourselves, techniques are daily being created to parse through and analyze the surplus of data coming from the devices we use all around us. With the huge shift in interest towards making smarter and more efficient technology, there comes a need for Machine Learning (ML) algorithms that can quickly make precise predictions. Among the many categories of ML, Supervised Learning techniques are used when there is a subset of data that has labels to accompany it. In scenarios like this, the labeled data is used to create and train a model that can be used to predict labels for subsequent data sets. K-Nearest Neighbor (KNN) is a relatively lazy Supervised Learning technique that can be used to classify data objects into preexisting labels.

Previous research has shown that utilizing KNN can be a very successful technique in image object detection. While KNN is preferred for it's simplistic and intuitive nature, it is incredibly computationally expensive and produces higher error as k approaches infinity [4]. There are a plethora of tools that can be used to lessen these problems, especially in the realm of image detection. When KNN was applied to identifying reoccurring objects in images that were multi-labeled, KNN outperformed many alternative algorithms [1]. Often KNN is used with support of other statistical techniques, like Principal Component Analysis, for object recognition and works well with images because it can reduce the dimensions of the very-large image data [2]. An alternative to data reduction techniques would be identifying local features. Research that focuses on identifying local features, like the work done in this paper, work with KNN in combination with techniques like Scale Invariant Feature Transformation (SIFT) and Speeded Up Robust Features (SURF), which use key points to identify objects and train the model [3].

Since the literature was very positive with respect to KNN techniques for object identification in images, I tested my own KNN algorithm using Euclidean Distance and taking advantage of neighboring pixels.

*This work was not supported by any organization

B. Richardson is with the Human Centered Computing Department, University of Florida, Gainesville, Florida 32611, USA richardsonb at uflorida.edu

II. Implementation

A. Identifying Red Pixels

The first step in the process is to identify the criteria for determining what a red pixel is. In this step, I used cross-validation of KNN to determine how many training points the validation set should look at to determine the respective label.

The training data was composed of the original red locations that were given in the 'ground_truth.npy' file and an equal set of randomly chosen indices that were a safe distance (at least 20 units) away from the ground truth coordinates that would be considered not red. This training data was split into training and validation data using Python's train_test_split function. The split was made 5 different times with 8 different odd K values. The optimal K was chosen based off the highest average accuracy and the lowest variance across the different splits. Only odd K values were used to avoid even splits.

B. Identifying Red Spaces

Once a technique had been made to determine, based on RGB, what a red pixel is, the next step was determining how large a car should be. Therefore, with the optimal K found in the algorithm above, another cross-validation was completed, except the varying parameter was the number of neighbors (N) that were considered a part of the car.

The training data was composed of the ground truth coordinates, all neighbors within an N distance away from ground truth coordinates, and an equivalent number of random coordinates that was a safe distance away from the ground truth coordinates. The data was split 5 times with 5 different N distances. The chosen N had the highest accuracy and the lowest variance.

C. Creating the Model

Once the optimal parameters were found, the training data (which was composed of the coordinates from the ground

truth, all neighbors N distance away from the ground truth, and an equal number of random points) was used to create the model. While the training data consisted of RGB values, the training labels consisted of 'red' or 'notRed' labels.

D. Testing the Model

Using a subset of the test image, the model was used to predict where the red cars were located. The testing functions looped through the image, looking at every N pixel instead of every pixel to reduce run-time. Each pixel was tested with the KNN model to see if it was red. If it returned red, all neighbors N-distance away from the pixel was tested with model to see how many neighboring pixels also came out to be red. If at least 50% of the pixels around it, came out to be red, this pixel was added to the array as a point in a car.

III. Experiments

There was a plethora of experimentation done to find the optimal parameters, constructing the training data sets, and testing the model to get good results.

A. Identifying the Best Algorithm

When it came to determining which algorithm to use, there had to be a decision between the Supervised Learning techniques that were used in class. The automatic choice seemed to be between KNN and Probabilistic Generative Classifier (PGC). Both of these techniques were tested using a collect of 56 points, 28 were the ground truth points and 28 were randomly chosen points at least 20 units away from the ground truth. With this rather small training set, cross-validation was done on both techniques. KNN was getting extremely high values, between 80 - 95%. On the other hand, PGC was consistently getting low values. This made sense considering that the R, G, and B values should not be taken for their face value, but in relation to each other. PGC's use of the mean and variance was not useful for this problem. Therefore, KNN was to be used.

B. Identifying the Optimal Parameters

The two parameters under consideration were the K value for KNN analysis and the N value to determine how many neighbors should be added to training data and tested to confirm they were part of a car.

Firstly, the optimal K was found using cross-validation. The data was split several times and in each split, 8 different K values were tested to see which one returned the best accuracy. As Fig. 1 shows, a K value of 7 was shown as optimal with the highest average and the lowest variance.

Since the distribution of incorrect predictions across the red and not red were not substantial, as can be seen in Fig. 2, the chosen parameter seemed to be both stable and not weighted to either label.

When determining the optimal N, a very similar technique was used. Cross-validation with the optimal K was run on training sets that included a ranging number of neighbors for each ground truth.

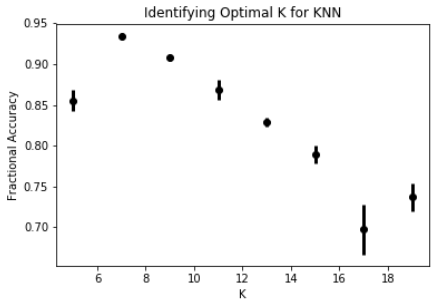

Fig. 1. Cross-Validation results when varying the K in KNN testing. The averages were plotted along with variance ranges.

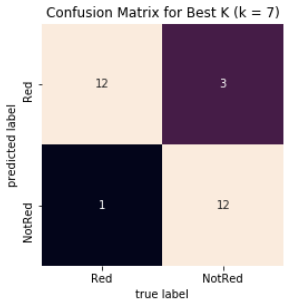

Fig. 2. Predicted vs. Actual Labels for optimal K of 7.

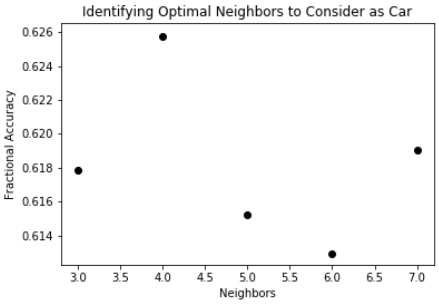

Fig. 3. Cross-Validation results when varying the N-number of neighbors considered as part of the training data. The averages were plotted along with variance ranges.

The results from Fig. 3 depict the averages across the varying N values and their corresponding variances. As can be seen, there was very little variance, which shows stability regardless of how the data is split. We can see that all the averages dropped from where they were in the optimal k cross-validation due to the increasing training set size.

The optimal N from the cross-validation analysis was 4. This meant that from the center of the car, all pixels 4 pixel units away could be considered a part of that car.

C. Identifying the Threshold for Confirming Red Car

Once the model has been created and pixels have been found to be red, the neighborhood of those red pixels are tested to see what percent of those pixels are red. A threshold had to be made for percentages of red pixels. When a threshold of

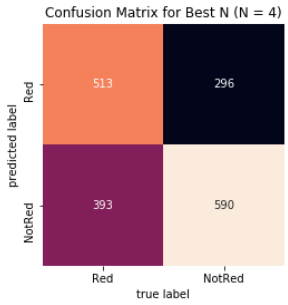

Fig. 4. Predicted vs. Actual Labels for optimal N of 4.

75% was used, few cars were ever returned. When a threshold of 50% was used, there was a more accurate return.

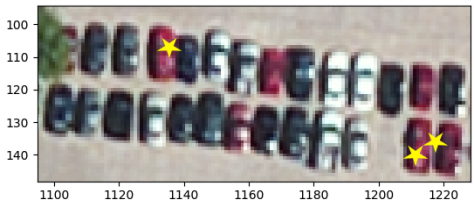

Fig. 5. One subset of image that was used for testing. Location of where cars were identified is starred.

From the results in Fig. 5, red cars were identified when the threshold was lowered to 50%. When the threshold was set to 62%, only one of those stars showed. There, of course, was many cars not detected, but the detection of red cars shows that this algorithm is working to a degree. The testing was done five different sub-images and returned very similar results in each.

IV. Conclusion

From the results and the process of creating the algorithm, many things became apparently clear. Firstly, KNN is very computationally expensive. Since no 'model' is made using this algorithm, but instead all testing sets are being compared to all training sets, more time is spent testing than should be, especially in an image.

With the intention of reducing the computation time, by considering only one car can exist in a N*N space, steps were taken every N pixels instead of every pixel. This did reduce the run time substantially, but considering there were 36mil pixels, it only reduced it by a fraction. Techniques that reduced dimensions or used parallel computing techniques would have been a lot more efficient.

Based on the results, richly red cars were more detectable than darker or lighter red cars. The majority of the predicted red cars are actually red cars. This means the model was probably too restrictive. If I had expended the ground truth to be more than 28 coordinates, the model might have detected more cars.

In conclusion, KNN is definitely a recommended technique when it's used in combination with reduction techniques. As a stand-alone technique, it would be better for smaller data sets, unlike the one here.

References

[1] M.-L. Zhang, Z.-H. Zhou, ML-KNN: A Lazy Learning Approach to Mult-label Learning (Periodical Style), Pattern Recognition, Vol. 40, Iss. 7, July 2007, pp. 2038-2048.

[2] R. Muralidharan, Object Recognition Using K-Nearest Neighbor Supported by Eigen Value Generated From the Features of an Image (Periodical Style), International Journal of Innovative Research in Computer and Communication Engineering, Vol. 2, Iss. 8, pp. 5521-5528.

[3] G. Amato, F. Falchi, On kNN Classification and Local Feature Based Similarity Functions (Conference Paper Style), Agents and Artificial Intelligence, January 2011, Communications in Computer and Information Science, Vol. 271, pp 224-239.

[4] J. Gou, L. Du, Y. Zhang, T. Xiong, A New Distance-weighted K-nearest Neighbor Classifier (Periodical Style), Journal of Information and Computational Science, Vol. 9, Iss. 6, 2012, pp. 1429-1436.