Project 00: Masked Averaging KNN Performance in the Classification of Red Cars

Ross Spencer, *(not part of the) IEEE*

***Abstract*—This project's aim was to find all red cars in an image. Given satellite imagery with labeled ground truth, I've taken 2 approaches based off the K Nearest Neighbors (KNN) algorithm to classify data. This paper is an attempt to use RGB values and the Minkowski metric with p=3 to classify a pixel as belonging to a red car or not. However, in the end, while the averaging gave a modest increase in accuracy on a localized validation set for small K, neither approach yielded usable results when scaled to a much larger testing file.**

***Index Terms*—Computer Vision, Machine Learning, Statistical Learning**

I. Introduction

How do we detect all the red cars in satellite imagery? In this project, I present 2 approaches to determine whether the object at an (x,y) coordinate in an image is a red car or not. This is a classification problem with 2 classes: red car, and not red car.

There is a fair amount of the literature devoted to using KNN to classify objects. While I initially considered switching color spaces to HSV or some other color space to see how that affected things, Cotrina et al., 2018, and Li et al., 2016, showed that sticking with RGB did as well or better than the other common color spaces typically using in computer graphics.

As far as which distance metric to use to determine how close each data point's nearest neighbors were, I referred to Mahama et al., 2016, in my decision to use the Minkowsky metric for my KNN, taking the cube root of the sum of cube error in the RGB channels.

Keeping the rest of my implementation relatively vanilla up until here, I noticed that in the ground truth data provided, sometimes the pixels were darker or lighter than the surrounding ones, and that even though the pixel belonged to a red car, it wasn't very representative of the color of the car as a whole.

II. Implementation

In order to compare my idea of averaging the data points' surrounding RGB values to a more standard approach, this paper proceeds with 2 approaches.

The first approach is a simple implementation of the KNN algorithm, provided by the scikit-learn package, using the RGB values of each pixel in a given ground truth array as its feature. This implementation uses the aforementioned Minkowsky metric, which turns out to be the default for scikit-learn's KNN in 3 dimensions to begin with.

The second approach is a very similar implementation of KNN that, for each (x,y) pair, instead takes the RGB values in a 5x5 mask around the associated pixel in the image and then averages them to smooth out the color values and dampen variability.

After some cross-validation, I decided based off my own heuristic observations that using 3 or 4 neighbors provides a good balance of accuracy without overfitting, especially since there are only 28 red cars in the ground truth to work off of. As such, both implementations utilize K=3 for classification.

The averaging algorithm is as follows:

```
AVG(X):
x ← [X[0] – 2, X[0] + 3)
y ← [X[1] – 2, X[1] + 3)
for i in x:
for j in y:
   r = r + image[x][y][0]
   g = g + image[x][y][1]
```

This work was supported in part by Chipotle and Starbucks for giving R. Spencer brain power and will to live.

R. Spencer is with the Machine Learning & Sensing Laboratory, and the Computer Science and Mathematics Departments at the University of Florida.

```
        b = b + image[x][y][2]
    r = r / (x.length * y.length)
    g = g / (x.length * y.length)
    b = b / (x.length * y.length)
    Ret (r, g, b)
```

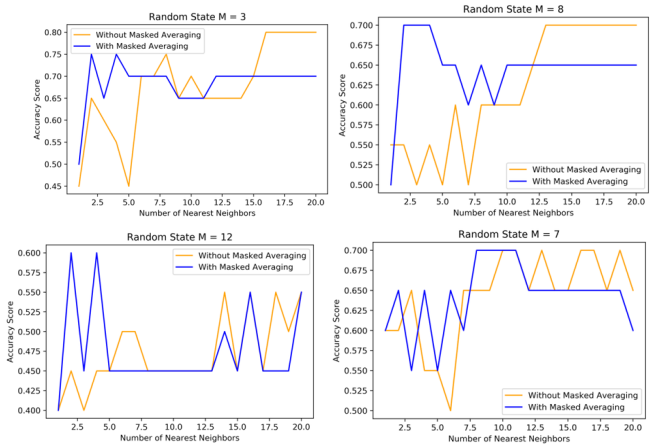
This is called on each coordinate X as a form of preprocessing before it is passed into our KNN training set.

Beyond this, we also have a method of not only training the classifier on a dataset X, train(X), but a method that calls that to train its classifier and then finds the predicted red cars on a window of an image. This method, findAllRedCars(Y, X, x1, x2, y1, y2, stepsize), takes in an image Y and a training dataset X, alongside parameters to control the window to reduce the time needed to run examples and also to allow the user to get a closer look at the points generated.

The window is controlled by limiting the x axis to (x1, x2) and the y axis to (y1, y2), with the step size allowing the user to control how many pixels to jump before looking for the next car. In particular, this is in part to help alleviate the overlapping of the kernel on cars, and also to prevent multiple-counting of cars that should be associated with multiple pixels. Some cars are 10 pixels long and 5 pixels wide, whereas others are 5 long and 10 wide and others altogether are slanted diagonally, making it hard without edge detection to avoid counting each red pixel as its own card.

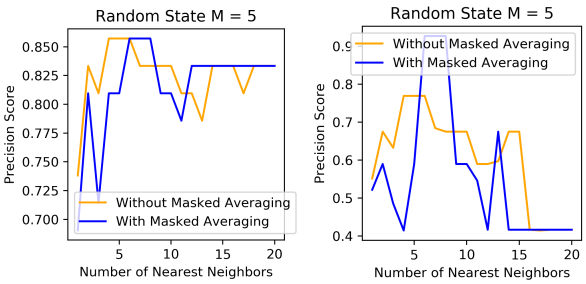
III. Experiments

In order to test my classifier, I first used cross-validation on my testing data to improve accuracy within that dataset. I split my data into a partition where 33% was used for testing and the remaining 67% was using for training. At first, I'd decided to use a bunch of randomly generated random points as my not-red-cars "ground rumor," seeing as the probability of each randomly-generated data point being at the location of a red car is effectively zero, but trying that resulted in the large number of non-red-cars overshadowing the meager 28 red cars in the ground truth array given with the project assignment. My classifier simply predicted that everything wasn't a red car. As a result, I went through and adjusted my ground rumor array by hand-labeling 32 different variations of features throughout the training image. Even still, my accuracy upon validation was fairly low, with most random states of the data split yielding around 70% accuracy.

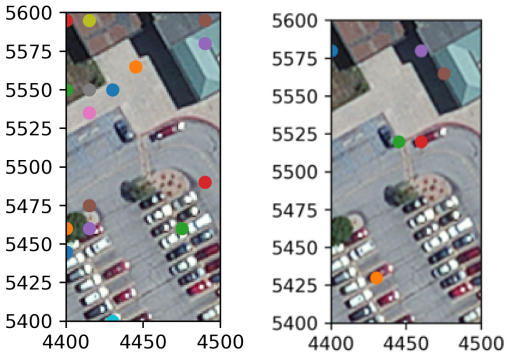


However, as can be seen in the figures above, the masking actually showed a modest improvement over traditional KNN for small K, but for large K ended up doing worse than traditional KNN, likely because it hindered overfitting.

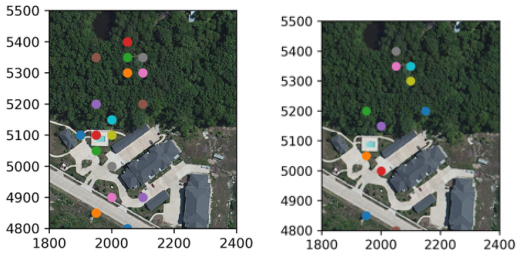
From here, I decided to test my precision. The implementations behaved about the same using micro-averaged precision and the masked averaging did slightly better in the macro average as shown in the two graphs below (micro on the left, macro on the right).



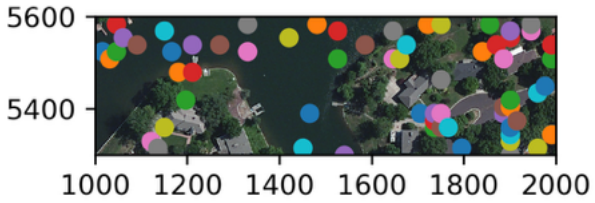
When it came to actually plotting the results of the trained classifier on either the training image or the testing image, however, both methods faltered completely. The masked classifier seemed to remove some of the noise in some cases, as shown in the figures below, with non-masked on the left and masked on the right.



However, clearly both missed a lot of red cars, and saw a lot of red cars where there were none. This is further exemplified in these forests that one would think had many invisible red cars.



Worse yet, for any area in the testing image that didn't seem well-represented in the training image and thus wasn't hand-labeled by me in the ground rumor array, it was a mess for both implementations:



IV. Conclusions

I really enjoyed this project as a whole. I felt like a mad scientist throughout all of my coding and experimentation, but I think I learned that KNN with such a small sample size is unreliable on the noisy, real-world data here, especially since the image is 6500 by 6500. I also think that what was happening resulted from the distance metric I chose, as it weights all 3 component channels of the RGB value equally. Hence, if one pixel had (x,y,z) as its RGB tuple and another had (y,x,z) as its RGB tuple, they probably looked the same to the KNN. However, trying to change to HSV for some reason made my red values appear green (around 180º), so I stuck with RGB.

A more sophisticated approach would probably be a way to include variance around a pixel or edge detection, but while I was initially worried about red roofs looking like red cars, so too did green lakes to my classifier.

References

[1] C. Cotrina et al., "Using machine learning techniques and different color spaces for the classification of Cape gooseberry (Physalis peruviana L.) fruits according to ripeness level." *PeerJ Preprints*, March 14, 2018. https://doi.org/10.7287/peerj.preprints.26691v1

[2] H. Li et al., "Identifying blueberry fruit of different growth stages using natural outdoor color images." *Comput. Electron*, Agric., 106:91–101.

[3] S. Mahama et al., "Choice of distance metrics for RGB color image analysis." *Society for Imaging Science and Technology*, February 15, 2016. https://www.ingentaconnect.com/contentone/ist/ei/2016/00002016/00000020/art00036