Tucker Mullens
CSC 391
Professor Pauca
April 18, 2019

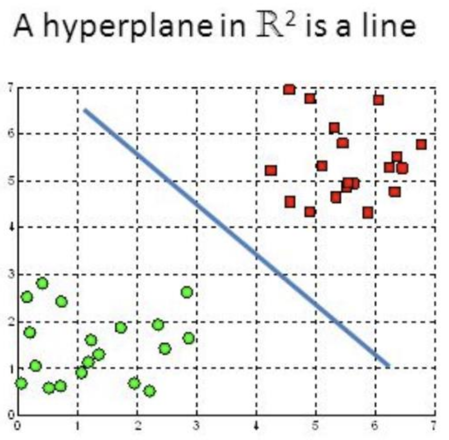
CSC391 River Classification Project

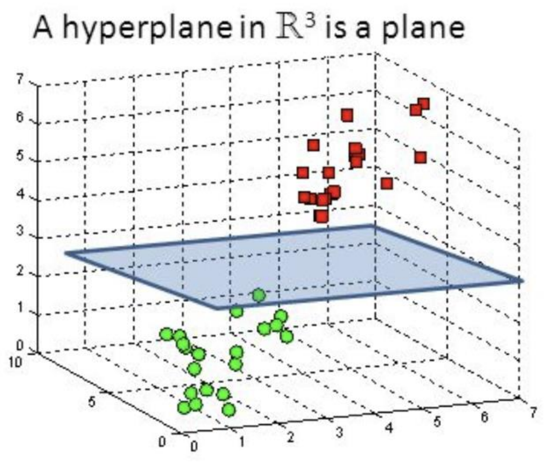
Introduction

The issue I am solving in this project is a binary classification problem. Using training data images of two types, aerial shots including rivers and aerial shots not including rivers, I will train a classifier to distinguish between "river" and "non-river" input test images. HOG Feature extraction was used in combination with an SVM classifier. In this continuation of the first assignment, I will experiment with SVM classification and the different parameters that can be edited to alter the efficiency of the its predictions.

SVM Overview

A Support Vector Machine classification model is a supervised classifier that builds a line or hyperplane in multidimensional to separate two classes for identification. The ultimate goal is to find a plane that has the maximum margin or distance between data points of both classes. Maximizing this distance provides reassurance that future predictions can be classified with a higher chance of success.





Implementation

The experiment was implemented in Python by first creating a dataset out of the provided images. The data images from the previous assignment were changed from their original dimensions of 300x300 pixels to 100x100 pixels since classification models typically fair better with images of a smaller size. Labels were then generated by separating the images into classes of “river” and “non-river” images. Excerpts from the code are shown below:

```
#Loading Images and Labels into training_data
DATADIR = "RiverImages"
CATEGORIES = ["river", "noriver"] #0 for river and 1 for noriver

training_data = []

for category in CATEGORIES:
    path = os.path.join(DATADIR, category)
    class_num = CATEGORIES.index(category)
    for img in os.listdir(path):
        img_array = cv2.imread(os.path.join(path,img), cv2.IMREAD_GRAYSCALE)
        training_data.append([img_array, class_num])
```

The HOG features were then calculated for each of the images in order to be eventually fed into the SVM classifier.

```
#Creating the hog features for the images
for sample in training_data:
    fd, hog_image = hog(sample[0], orientations=8, pixels_per_cell=(10, 10),
cells_per_block=(4, 4), block_norm='L2-Hys',
                        visualize=True)
    hog_images.append(hog_image)
    hog_features.append(fd)

hog_features = np.array(hog_features)
```

The features and labels were then moved into a numpy hstack so they could be shuffled to ensure that the river and non-river images were evenly distributed among the dataset when training and testing the classifier.

```
#Moving hog_features and labels into a data_frame and shuffling it
data_frame = np.hstack((hog_features, Y))

np.random.shuffle(data_frame)
```

The SVM classifier was generated using sklearn’s svm.SVC mehod call.

```
#Creating the SMV classifier
clf = svm.SVC(C=0.001, gamma=0.1)
```

The data was then partitioned into training and testing data, the model was fitted, and the predictions were generated.

```
#Partitioning data_frame into training and testing data
x_train, x_test = data_frame[:90, :-1], data_frame[90:, :-1]
y_train, y_test = data_frame[:90, -1:].ravel(), data_frame[90:, -1:].ravel()

#Fitting model to training data
clf.fit(x_train, y_train)
```

```
#Generating predictions for test data
y_pred = clf.predict(x_test)
```

The output of the predictions was formatted in the following way, producing a zero value predicting the presence of a river and a one value predicting a non-river image. The ratio of correct predictions to incorrect predictions, or accuracy was output as well. Sample output:

Results:
Predicted: 1.0 Actual: 1.0
Predicted: 0.0 Actual: 0.0
Predicted: 1.0 Actual: 1.0
Predicted: 0.0 Actual: 0.0
Predicted: 0.0 Actual: 0.0
Predicted: 1.0 Actual: 1.0
Predicted: 1.0 Actual: 1.0
Predicted: 0.0 Actual: 0.0
Predicted: 0.0 Actual: 1.0
Predicted: 0.0 Actual: 0.0
Accuracy: 0.9

First Attempt

Originally starting the project with the same data collected from the previous assignment, 25 images containing rivers and 25 images not containing rivers were used to train the SVM classifier. The parameters for the SVM were left on the default values set by the sklearn package, and the following parameters for HOG feature extraction were used:

```
hog(sample[0], orientations=8, pixels_per_cell=(20, 20), cells_per_block=(2,
2), block_norm='L2-Hys', visualize=True)
```

The program appeared to produce decent results at the first run, producing an accuracy of 0.8 in correctly predicting the test data.; however, running the program more than once appeared to produce issues. There were many consecutive runs of the program in which the classifier only predicted all zeros or all ones usually resulting in low accuracy rates in the range 0.3-0.5. There was uncertainty in why exactly this was occurring to the classifier and it usually took several runs of the program before it made another decent prediction that did not consist entirely of one class.

Second Attempt

Arbitrarily guessing that the problem may be solved by altering the parameters for the HOG feature extraction, the pixels_per_cell parameter was changed to (10, 10), thus increasing the length of the feature vector for each image. The thought process followed that the more features that could differentiate the two categories, the more accurately the classifier would be trained to predict them apart. Unfortunately, running the program still produced outputs of all zeros or all ones predictions, however a good run of the program produced an accuracy of 0.9 in correctly predicting the test data. The one-sided predictions were still occurring with the same frequency however.

Third Attempt

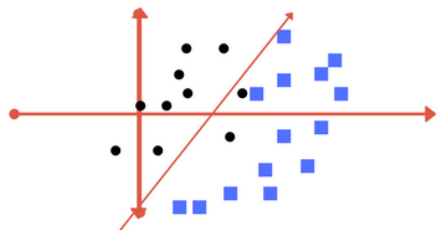
Further altering the parameters of the HOG feature extraction, the cells_per_block parameter was changed to (4, 4) which means that 16 of the 10x10 pixels are normalized together in a block, possibly allowing for the detection of contrast over larger regions than previously specified. Running the program with this alteration produced a resultant accuracy of 1.0 or all correct predictions for one run of the program. The issue of predicting all within one class still persisted however, giving reason to look for explanations in other parts of the experiment setup.

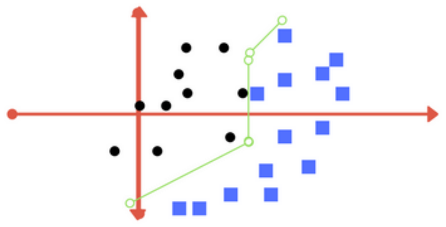
Fourth Attempt

Performing some basic research on why exactly the issue could be occurring, a common answer attributed it to uneven distribution of the classes of data used. In this setup, there were equal numbers of river and non-river images however. The smaller size of the dataset could possibly cause the issue since if a long run of elements of the same class occurs in the training set due to shuffling, this may lead the classifier to lean towards predicting only one class. Therefore, more data images were created from the provided satellite images, doubling the size of the dataset to 50 river and 50 non-river images. The partition was therefore changed to 90 images as training data, leaving 10 for test data. Running the program however, simply seemed to produce similar results as before, still having the same issue. Good runs of the program produced accuracies of 0.7-0.8 predictions, however, the problem was still occurring with the same frequency.

Fifth Attempt

Continuing further research into the nature of the problem, fine-tuning parameters for the SVM classifier were suggested as a possible solution to the plight of the experiment. The svm.SVC() method call from sklearn has two parameters of interest, “C” or “the penalty parameter of the error term” and “gamma” or the “kernel coefficient”. The C parameter essentially controls the misclassification of the data. It decides between the trade off of correct classification of training data against the maximization of the decision function’s margin. Thus, larger C values mean a smaller margin will be accepted if the decision function is better at classifying all training points correctly, while a lower C value creates a larger margin and therefore a simpler decision function at the cost of a possible loss of training accuracy. The left graph below shows what an SVM hyperplane may look like if a lower C value is used as opposed to the right plane which displays the effects of a possible higher C value.





Gamma according to sklearn defines “how far the influence of a single training example reaches, with low values meaning far and high values meaning close.” Unsure of how to decide the values for these parameters, optimization was performed using Gridsearch, which is used to find the optimal hyperparameters of a model. The Gridsearch code used is shown below:

```
def svc_param_selection(X, Y):
    Y = Y.ravel()
    Cs = [0.001, 0.01, 0.1, 1, 10]
    gammas = [0.001, 0.01, 0.1, 1]
    param_grid = {'C': Cs, 'gamma' : gammas}
    grid_search = GridSearchCV(svm.SVC(kernel='rbf'), param_grid)
    grid_search.fit(X, Y)
    grid_search.best_params_
    return grid_search.best_params_
```

The function returned C = 0.001 and gamma = 0.1 as the best values to use as input for the SVM classifier. The idea was that a possibly lower C value may allow more lenience for images and would create a less rigid model where the classifier would predict outside of simply just one class. Inserting these parameters, the model appeared to produce more frequent appearances of decent results with accuracies sitting in the range of 0.7-0.9. Unfortunately, the original problem still persisted, still causing breaks in between good runs of the program.

Error Analysis

The constant appearance of predictions all within one class persisted throughout this experiment despite alterations of inputs and parameters and the error may be resulting from an overlooked aspect of the testing. There may simply be a lack of enough data to efficiently train the classifier, so it ends up simply predicting all within one class. A larger dataset may serve as a decent solution. An SVM classifier itself may simply be unsuited for this project and not the ideal choice for classifying this exact dataset. Despite any possible solutions, the presence of the issue must be recognized as a harm to this experiment and decrementing the possible validity of the predictions generated by the classifier.