Sattelite Ship Classification with Dimensionality Reduction via Python Machine Learning Models

Andrew Perez-Ledo
University of Florida
Gainesville, United States
andrewperezledo@gmail.com

***Abstract*— Machine learning is continuing to be employed for more complicated tasks which can result in large dimension sizes of data. In this report, we will cover the implementation of the scikit-learn Python library to classify images of ships taken from satellites while gauging the effects of dimensionality reduction on the image data.**

Keywords—F1 Score, Training, Inference, Dimensionality Reduction, Accuracy, Classifiers

I The Data

The dataset contains pictures of various bays taken by a satellite orbiting the planet. These pictures focus on the San Francisco and San Pedro Bays in California. There are 4,000 images, each one being 80 pixels by 80 pixels and colored (red, green, and blue). Scientists labeled each image as either having a ship in it or not having a ship in it. For an image to be labeled as having a ship in it, the full length of the ship should be in view of the image. Each pixel in the image corresponds to a real-world area of 3 meters by 3 meters. Refer to Figure 1.

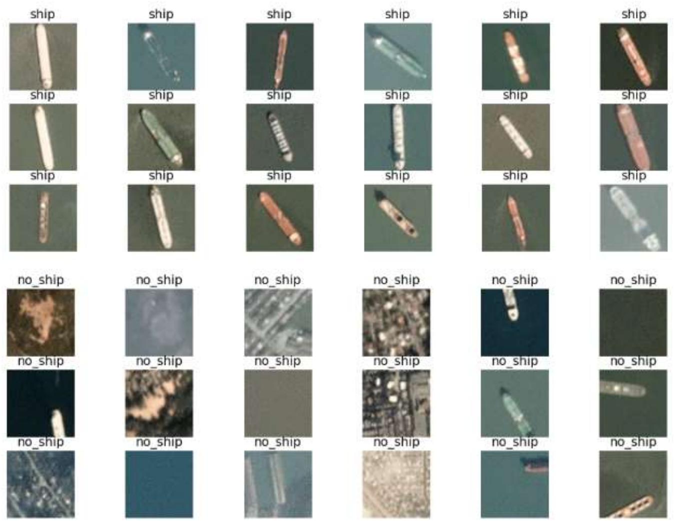


Figure 1, Images depicting "Ship" and "No Ship" Classifications

II The Goal

The end goal to achieve is finding the best overall model for the proper classification of images as either containing a ship or containing no ship as previously defined. This must be done by incorporating different methods of classification and dimension reduction in order to also provide a model with adequate training/inference times as opposed to a highly accurate model with extreme action times.

III Preparation for Model Creation

Before we construct our models, we must first manipulate the data received into a form that scikit-learn's objects can interact with. Therefore, we first separate the given four-dimensional array of 4000x80x80x3 into a training and testing set. After this, we must reshape the data into a two-dimensional array where the first dimension corresponds to each image and second dimension contains the RGB (red-blue-green) values for each pixel within the image.

IV model creation, training, and evaluation

Scikit-learn provides simple functions and classes for the creation and manipulation of machine learning models. We will use these along with matplotlib functions to visualize some aspects of our trained models.

V Classifiers

Our experiments will revolve around two chosen classifiers, the random forest, and the support vector machine. These classifiers were chosen due to their inherent differences in abilities. Random forests are ensemble learners which trains multiple smaller decision trees in parallel resulting in faster training/inference times for larger datasets with higher dimensions. On the other hand, support vector machines (SVMs) are known for their extremely high performance as a classifier at the cost computational power draw making them suitable for smaller datasets with lower dimensions. The pairing of the two has been chosen to provide further insights into which types of learners would be more effective and efficient with dimensions are high and when dimensions are reduced.

VI Trained Classifiers Alone

The first models trained were pipelines with just the classifiers with no dimensionality reduction preceded by a standard scaling of the data. The performance of these models will serve as a baseline for comparing the effect of dimension reduction.

Our random forest classifier, after hyperparameter tuning, had a training time of 22.12 seconds with an inference time of 0.07 seconds. Additionally, it had an accuracy of

95.25% and an F1 score of 93.41% (refer to Figure 2 for this model's confusion matrix).

The SVM, as expected, had a much longer computing time for hyperparameter tuning. Once the best hyperparameters were found, training and inference time were 13.54 seconds and 8.92 seconds respectively. It achieved an accuracy of 96.12% and an F1 score of 94.67%.

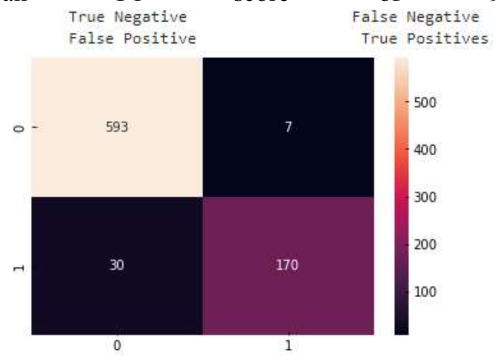
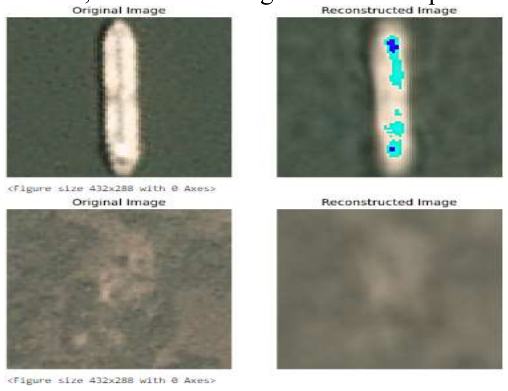


Figure 2, Correlation matrix of first model

VII Classifiers With PCA Dimension Reduction

Principal Component Analysis (PCA) is a dimensionality reduction technique that transforms high-dimensional data into a lower-dimensional space by capturing the most significant variance within the data. This is achieved by identifying a set of orthogonal features, called principal components (PCs), that explain the majority of the data's variability. In our work, we employed PCA to preprocess the data before training the same classifiers.

A common implementation of PCA includes using the number of components needed to maintain 90% of the data's variance. It was found that at least 104 components were needed to maintain this percentage. Figures 3 & 4 are reconstructed images of PCA reduction, and their average root mean squared error.


Original Image
Reconstructed Image
<Figure size 432x288 with 0 Axes>
Original Image
Reconstructed Image
<Figure size 432x288 with 0 Axes>

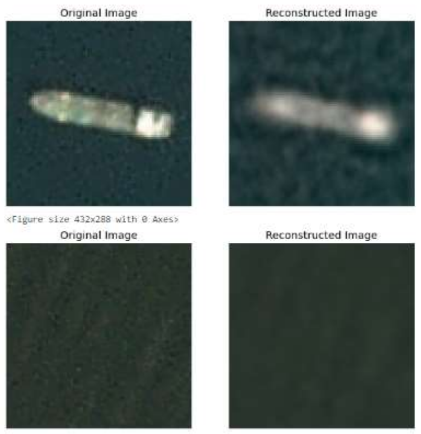


Figure 3, Reconstructed Images from PCA

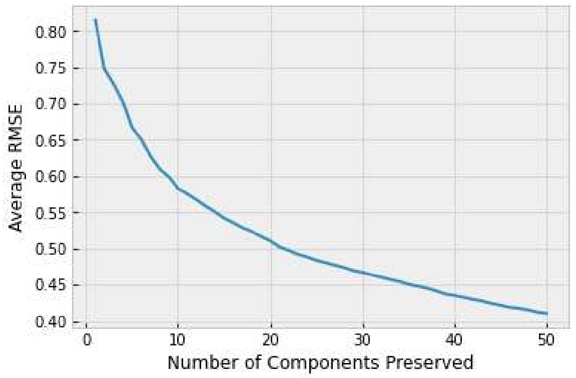


Figure 4, Plot of Average RMSE to Number of Components Preserved in PCA

Our models, implemented with PCA, provided relatively well performance in spite of the large reduction of dimensions. The random forest trained in 3.98 seconds and provided inferences in 0.10 seconds that had an accuracy of 95.12% and F1 score of 93.15%. The SVM trained in 4.59 seconds, inferred in 0.17 seconds, and resulted in an accuracy of 95.87% and F1 score of 94.35%. Between these two models, the "best" can be chosen, but it is subjective depending on what model preferences there are. In this case, we are willing to sacrifice a small increase in training time for a small increase in accuracy, so the SVM model will be considered the best.

VIII Classifiers With Manifold Learning

Manifold learning algorithms are another dimensionality reduction technique that uses different assumptions to capture different relationships between data points, especially non-linear relationships. We utilized two manifold learning algorithms, Isometric Mapping (Isomap) and Locally Linear Embedding (LLE) instead PCA for these models. Isomap explores the shortest paths within the data's manifold while LLE represents each datapoint as a linear combination of its nearest neighbors. The isomap models outperformed the LLE models with the same classifiers in every metric. However, it is important to note that both classifiers performed worse on every instance a manifold learning algorithm was used.

For example, the random forest incorporating Isomap managed a training time of 13.74 seconds, inference time of 1.66 seconds, accuracy score of 92% and F1 score of 89.03%. Furthermore, the SVM with Isomap produceda training time of 12.01 seconds, inference time of 1.749 seconds, accuracy of 92.87% and F1 score of 90.14%.

Regarding the models with LLE, random forest took 15.02 seconds to train, 2.046 seconds for inferences, accuracy was 92%, and the F1 score was 88.79%. SVM trained in 9.97 seconds and provided inferences in 2.14 seconds which had an accuracy of 92.5% accuracy and an F1 score of 89.45%.

These models with different manifold learning algorithms gave comparable results despite their differences in computation. An increased understanding of how these work can be achieved by visualizing the the first two components of both algorithms when ran, refer to figure 5 and 6.

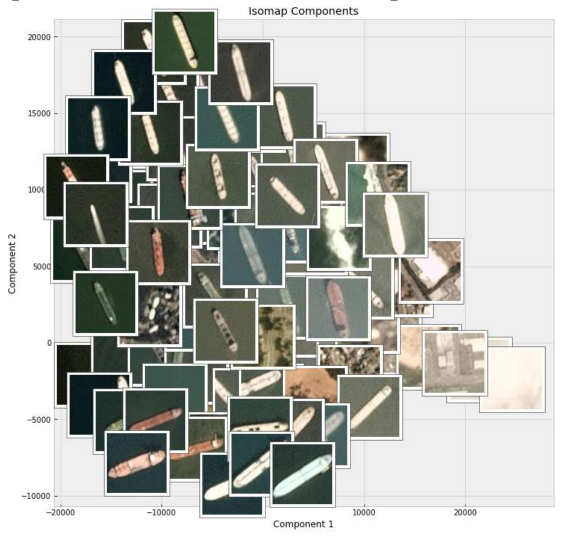


Figure 5, Visualized plot of first 2 Isomap Components

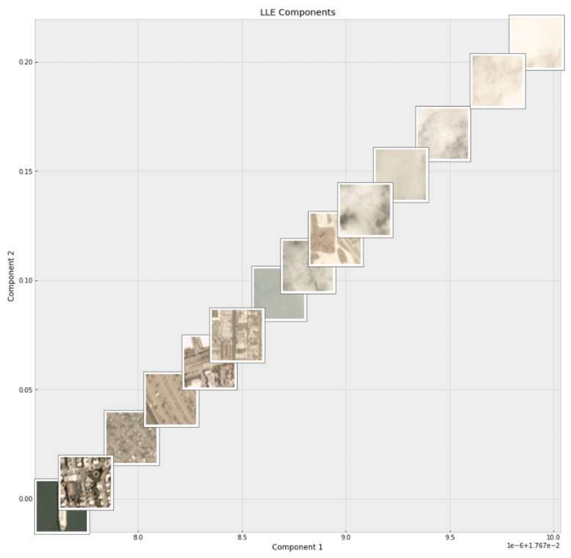


Figure 6, Visualized plot of LLE's first 2 components

The visualization for the isomap reveals patterns in their components. The first component, the x-axis, seems to correlate with the brightness of the overall image. More images containing dark sea water are on the left while brighter images of land and buildings are on the right. For the second component, the y-axis, appears to vary in the orientation of the ships in the images. More horizontal ships appear towards the bottom while more vertical ships are shown at the top of the plot.

The LLE's plot is more peculiar. All images are in a straight line like the plot of the function f(x) = x. The image at the plot's origin is perfectly vertical ship. The rest of the images as they grow further from the origin lose more and more varying objects within the image as to change from an image with streets and buildings to images of almost solid single color. One thing it could also be considering is the brightness of images as the furthest image from the origin is the brightest. Based on this, the first two components could represent the complexity of the image and its brightness.

IX Best Model

The best model of all the combinations of dimension reductions and classifiers has been the SVM classifier with PCA dimensions reduction. It provided the highest accuracy out of any model with dimension reduction and had a smaller training time than its non-dimension reducing counterpart.

X Misclassifications

Throughout the experimenting of all these constructed pipelines, misclassifications were always present, and always showed consistency. All models, even the best, consistently misclassified images of "no ship" images as "ship" images. This has been prevalent on all confusion matrices plotted for all models where the category of false positives comprised of a large majority of the total number of misclassifications. This phenomenon is most likely due to the abundance of ship-like objects in images. This discovery is easily made when accessing misclassified images as most of them are "no ship" labeled images.

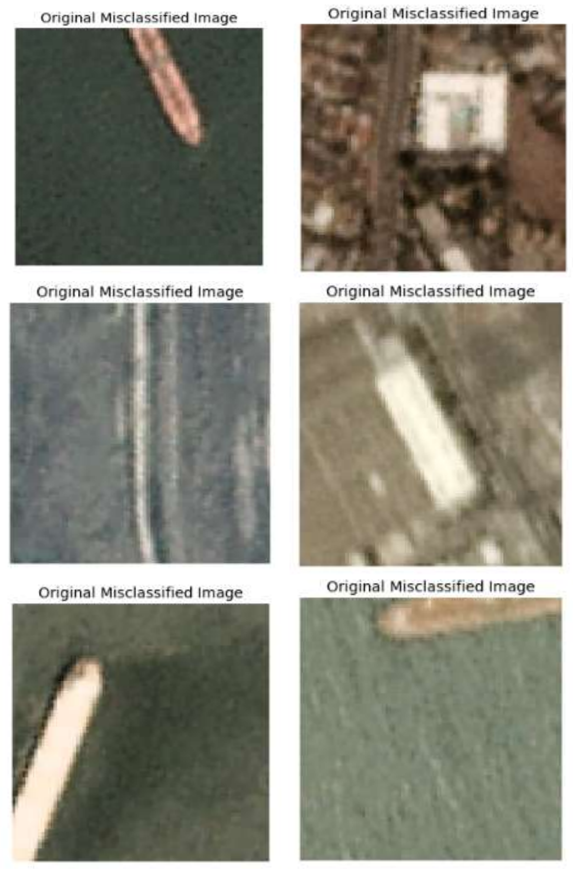

Original Misclassified Image
Original Misclassified Image
Original Misclassified Image
Original Misclassified Image
Original Misclassified Image
Original Misclassified Image

Figure 7, Misclassified Images

To move forward with our model creation and prevent common causes of misclassification such as these, several steps can be taken. Before, however it is important to note that the dataset did contain a misbalance of classes being 1000 for "ship" images and 3000 for "no ship" images. Even though the data was stratified upon being split, there could still be an underlying learning bias in effect. One step can be to apply different weights to the two classes, placing a higher emphasis on the negative class to learn them more accurately. This would effectively add a penalty to when the model misclassifies a "no ship" image. Another one could be experimenting with different hyperparameters such as the SVM's kernel. Additionally, creating a custom ensemble learner with multiple different learning systems could be useful.