Project Collaborators: Sean Emery, Hope Fowler, Travis Thein, Joey Sortino
Machine Learning
Professor K. Leonard
Monday, November 25, 2019

Classification of Coral Reef Recruitment: Naive-Bayes and Soft-Margin SVM

Classification Problem

This project involved an interrogation of the SVM and Naive-Bayes algorithms with a marine classification data problem posed by Professor Dr. Amber Stubler. Dr. Stubler's research is concerned with the marine community ecology of kelp and coral reef ecosystems, honing in specifically on Caribbean coral reef recruitment under different sedimentation regimes.

In order to document changes over time, Stubler placed tiles in specific areas in order to collect photographs of the tiles over the course of two and a half years. Pictures of the tiles were taken at the 6-month, 18-month, and 30-month intervals to track marine populations over time.

Dr. Stubler has asked us to help identify the various organisms in each photo. This project is important because Professor Stubler's methodology of tracing each organism by hand is a very lengthy, demanding, inefficient process.

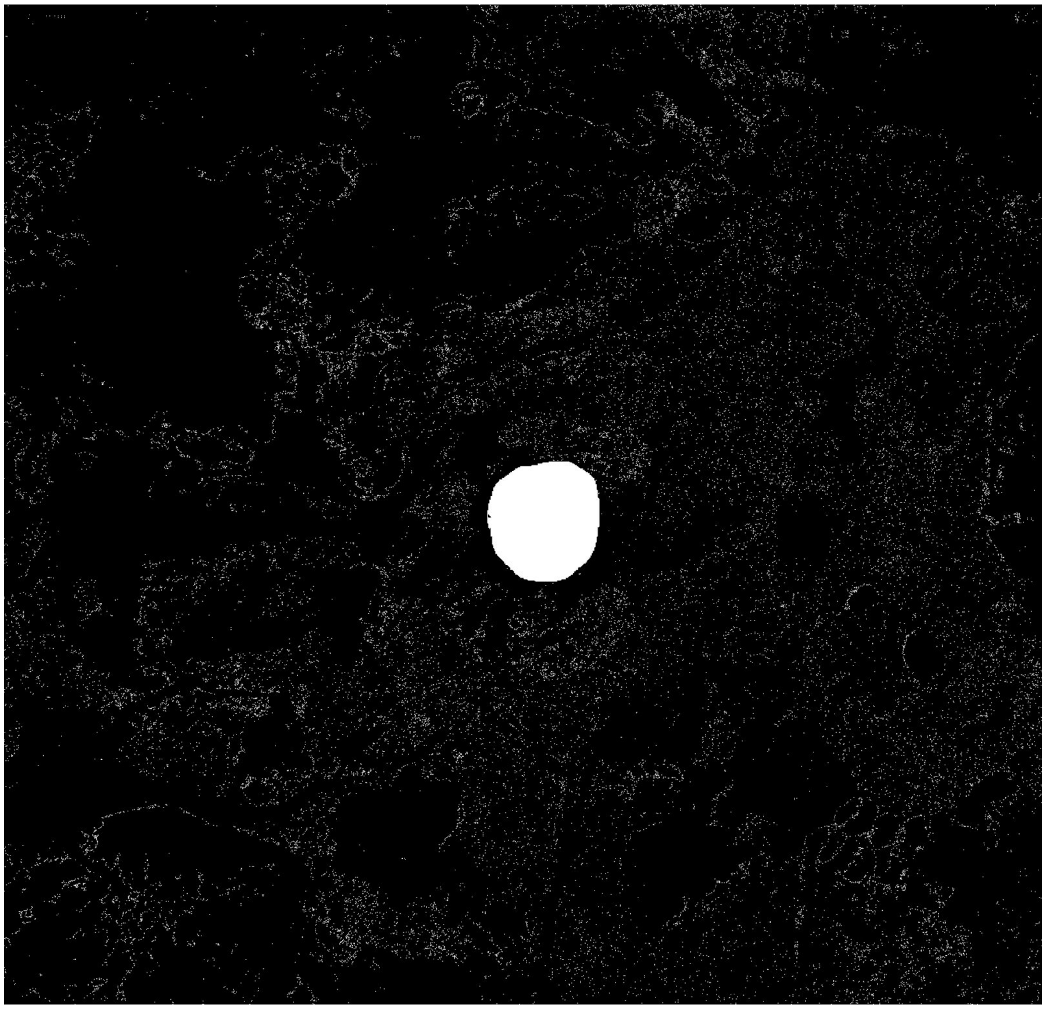
Dr. Stubler has given us a two-part dataset. The first part is a set of tile photographs, and the second is the same set of images with a purple highlighter over the central hole in each photo and a blue highlighter over all instances of coral in each photo. Professor Stubler also gave us an excel sheet outlining additional information about each of the images, such as location, year, month, depth, site (east/west), species, area, total_Area, and prop_Area.

Our goal is to attempt to classify the hole in unlabeled images by training both a Naive-Bayes model and a soft-margin SVM with the manually-labeled images.

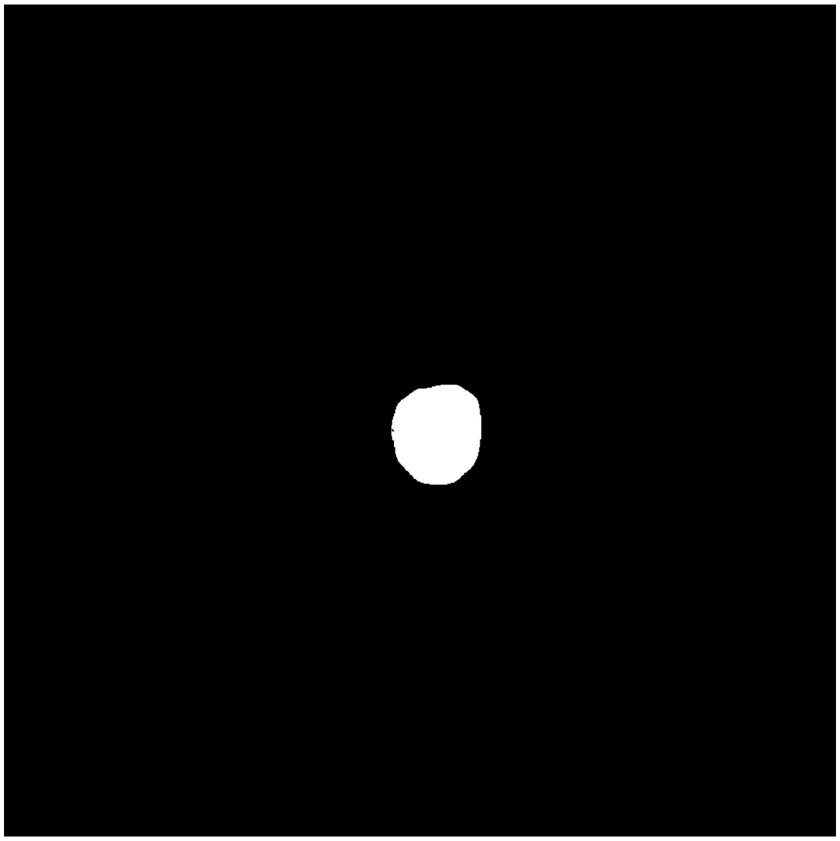
Data Preprocessing & Feature Engineering

Identifying the Highlighter

First, we needed a script that could identify the hole highlighter in each labeled image and derive "hole/not-hole" classification labels for each pixel in that images associated unlabeled image. To do this, at each labeled image we loop through every pixel and ask at each whether it is part of the highlighter or not. The highlighter seemed uniform in color between images, and so we first attempted to look for specific R, G, and B values that we manually extracted from one of the images. We quickly learned, however, that the highlighter color varies slightly between images, meaning that there was no key set of RGB values to look for. Therefore, to recognize the highlighter in each of the labeled images, we converted them to grayscale, in which the highlighter shows up as a uniform gray value of 84. The first part of our preprocessing script loops through the grayscale image, turns every pixel with the grey value of 84 white, and all others black, as pictured below.



The image is noisy because a number of pixels outside of the actual hole returned the gray value 84. To remove the noise, we used the MatLab's *bwareaopen* method, which takes a binary image as its input, then it removes components that are in smaller clusters than a set pixel count threshold and returns a new image with the components removed. Our threshold is set to 250.



This processed labeled image is now a classification key for its associated unlabeled image. When forming the training set with pixels from the unlabeled image, the proper “hole/not hole” label is derived from checking whether the pixel at the same location in the processed image is white or black respectively.

Feature Extraction

Because we are addressing a greater classification problem by summing individual pixel classification, we had to extract a set of pixel features. The only feature values readily available with this data were the three color values (red, blue, and green), and the location values (x and y coordinates). We knew that these five features would be relevant to our classification problem because the holes had consistent placement and relatively consistent coloring across the training images and so we used them for our feature set.

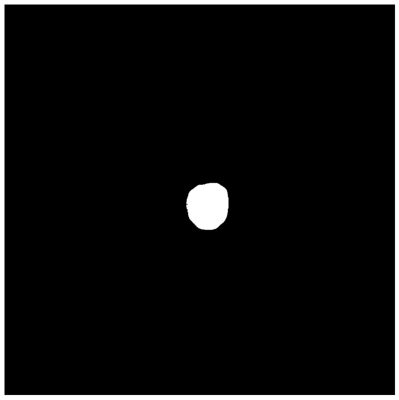
Assembling Training Set/Even vs Random Sampling

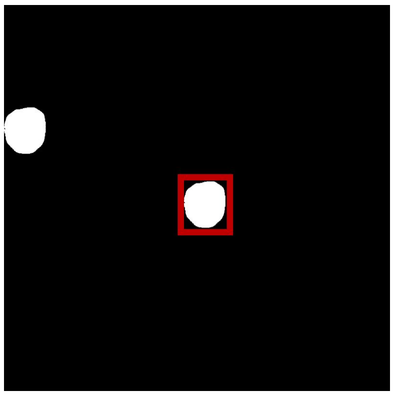
Our training set is a matrix with a row for each pixel and columns for the features (X-Coordinate, Y-Coordinate, R-Value, G-Value, B-Value), followed by a label column (1 for hole, -1 for not hole). We built this matrix by extracting feature information directly from each pixel used in the set and extracting the label from the associated pixel in the relevant processed image.

Had we used every pixel from each of our training images, this matrix would have been nearly 5 billion rows long and far out of the scope of this project. Instead, we tried two kinds of data sampling to achieve a more manageable amount of extracted data.

Even sampling (equal number of "hole" and "not-hole" training data), in our case, would add all pixels from the hole to the training set, and then take an identical cluster of pixels elsewhere in the image (a random location whose bounded box does not overlap with hole's bounded box) as non-hole data.

While even sampling became useful once we were working with the SVM, random sampling, in which we only used one-fifteenth of all pixels in each image, randomly chosen, was the preferred sampling method for Naive-Bayes (see Findings and Conclusion Section).


Original Cutout (Random Sampling)


Bounded Box (Even Sampling)

Visual Representation of Even Sampling

Data Tabulation: (Even)

Value	Count	Percent
-1	35772	50.00%
1	35772	50.00%

Data Tabulation: (Random)

Value	Count	Percent
-1	31847181	99.09%
1	292538	0.91%

Corrupt Images & Training/Test Split

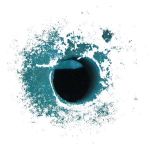
Of our set of 164 manually-labeled images, 14 of them were corrupt, either with white text boxes that overlaid the highlighting in the images or with the incorrect use of highlighting color, making them unfit for as training images from the onset. Because of this, we began working with our Naive-Bayes model excluding these 14 images from the training set. We decided to use this set of 14 images as our testing set since they were already random instances of the overall data set and were not available to be incorporated in the training set. However, we realized afterward that we only had qualitative data to analyze (the output images of the predicted hole isolated against a white background) and so we decided to manually correct the images so that we could obtain quantitative results. Having the corrected, labeled images for the test set meant that we could know the proper classification for each pixel in the test images and thereby afforded us empirical measures of model performance accuracy.

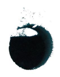
Naive-Bayes Implementation

Using the built-in MatLab functions, we were able to easily train a Naive-Bayes model using the command *fitcnb* and *fitcnb* with Cross-Validation enabled. We also used MatLab's *predict* method to obtain labels that we used in our confusion matrices and visual reproductions of what our model classified as a hole.

When we first were considering how to sample data for training Naive-Bayes, we prioritized having an non-hole pixel in the training set for every pixel that belonged to the same hole, thereby having a training set evenly split across the label. We implemented this idea with the Even Sampling method, which drew a bounded box around the center hole (to prevent us from sampling not-hole pixels within the actual hole). At this first step, the results were lackluster but unsurprising, as we had anticipated hole/non-hole data would not be linearly separable based on the location coordinates and RGB values alone (image dimensions ranged from 1500*1500 to 2000*2000).

Eventually, though, we tried the more conventional Random Sampling method. The results were impressive to an extent we thought we would find only by investigating non-linear solutions. Below is a sample of the difference in accuracy achieved by switching from even to random sampling.





Naive-Bayes with Even Sampling Naive-Bayes with Random Sampling

Having discovered that the random sampling method was far superior to even sampling for our purposes, we dismissed the even sampling code and moved on to implementing our Soft-Margin SVM.

Soft-Margin SVM Implementation

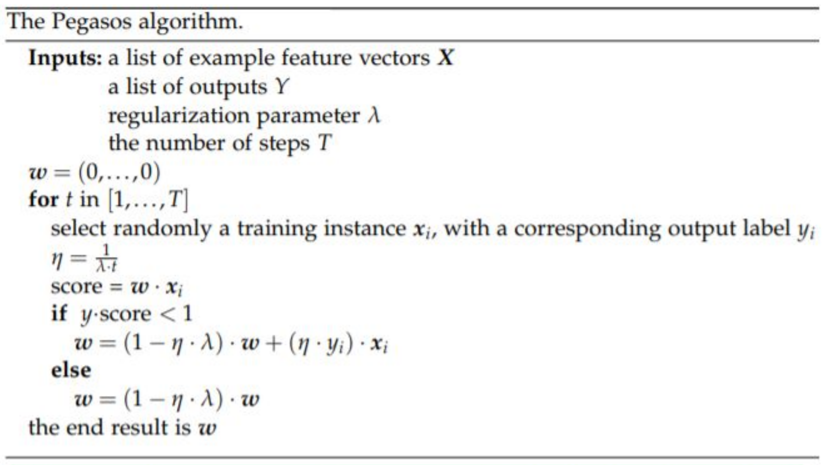
Implementing our Soft-Margin SVM proved to be the most difficult and time-consuming part of this project. Multiple days of work were dedicated to debugging the code that seemed entirely error-free. This entailed multiple instances of scrapping our progress and trying to implement different learning algorithms, to see if we would have any better luck. These algorithms included sub-gradient descent with both the stochastic and nonstochastic variants of PEGASOS (Primal Estimated sub-GrAdient SOlver), as well as attempting the 'kernel trick' with kernelized PEGASOS, coordinated descent with LIBLINEAR, and general quadratic programming, sans the Optimization Toolbox and MatLab's *quadprog* function that comes with it. Throughout this period of work, we were consistently attempting to train the SVM with the same data that had worked so well for the Naive-Bayes model.

In a last-ditch effort, we tried training the SVM with the even sampling dataset, which, again, had been previously dismissed, and the SVM was suddenly able to train and classify. In the end, it was not a coding error holding us up, but a data error. Our best guess is that the SVM was responding poorly to the extreme downsizing of the data. Given that the hole only makes up an average of 1% of each image, randomly sampling the images creates a training set with an almost negligible amount of actual hole pixels.

A few lessons we've taken away from this frustrating challenge:

- Never discard old code and always exercise proper version control. You never know when you might need something that you previously thought was useless.
- Trust personal coding competency and intuition and be open to looking for the problem in other pieces of the puzzle.
- What a confusion matrix is, and how to interpret them.

The algorithm we ended up opting for was the PEGASOS Stochastic Algorithm,



The Pegasos algorithm.

```
Inputs: a list of example feature vectors $X$
        a list of outputs $Y$
        regularization parameter $\lambda$
        the number of steps $T$
$w = (0,\dots,0)$
for $t$ in $[1,\dots,T]$
  select randomly a training instance $x_i$, with a corresponding output label $y_i$
  $\eta = \frac{1}{\lambda \cdot t}$
  score = $w \cdot x_i$
  if $y \cdot$score $< 1$
    $w = (1 - \eta \cdot \lambda) \cdot w + (\eta \cdot y_i) \cdot x_i$
  else
    $w = (1 - \eta \cdot \lambda) \cdot w$
the end result is $w$
```

Taken from https://svn.spraakdata.gu.se/repos/richard/pub/ml2016_web/a2_clarification.pdf

This Stochastic Sub-Gradient Descent Algorithm is obtained from taking the derivative of a Soft-Margin SVM that randomly samples observations from each row. For our actual SVM implementation, it was essentially a direct, one-to-one translation of the pseudocode above to MatLab's programming language.

Model Comparison

Average Case Confusion Matrices

	Naive-Bayes	***Soft-Margin SVM***
Random Sampling	**99.0013 0.2808** **0.3627 0.3552**	**99.2821 0** **0.7179 0**
Even Sampling	**97.5500 1.7320** **0.1191 0.5988**	**99.0327 0.2494** **0.6236 0.0943**
Random Sampling w/ Cross-Validation	**99.0013 0.2808** **0.3628 0.3552**	**X**
Even Sampling w/ Standardized Data (Trick)	**X**	**98.3406 0.9414** **0.2173 0.5006**

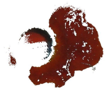
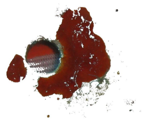
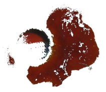
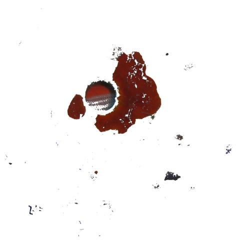
Worst Case Confusion Matrices

	Naive-Bayes	***Soft-Margin SVM***
Random Sampling	$\begin{matrix} 96.8075 & 2.4747 \\ 0.4626 & 0.2552 \end{matrix}$	$\begin{matrix} 99.0654 & 0 \\ 0.9346 & 0 \end{matrix}$
Even Sampling	$\begin{matrix} 95.6592 & 3.6230 \\ 0.0622 & 0.6556 \end{matrix}$	$\begin{matrix} 98.3427 & 0.9395 \\ 0.7178 & 0 \end{matrix}$
Random Sampling w/ Cross-Validation	$\begin{matrix} 96.8077 & 2.4745 \\ 0.4627 & 0.2551 \end{matrix}$	**X**
Even Sampling w/ Standardized Data (Trick)	**X**	$\begin{matrix} 95.8282 & 3.4540 \\ 0.1112 & 0.6066 \end{matrix}$

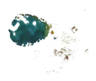
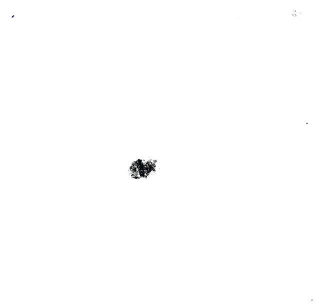
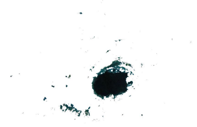
Best Case Confusion Matrices

RESULTS	***Naive-Bayes***	***Soft-Margin SVM***
Random Sampling	$\begin{matrix} 99.2030 & 0.1122 \\ 0.0608 & 0.6240 \end{matrix}$	$\begin{matrix} 99.4630 & 0 \\ 0.5370 & 0 \end{matrix}$
Even Sampling	$\begin{matrix} 98.9788 & 0.1453 \\ 0.1964 & 0.6795 \end{matrix}$	$\begin{matrix} 99.3291 & 0.0094 \\ 0.3873 & 0.2742 \end{matrix}$
Random Sampling w/ Cross-Validation	$\begin{matrix} 99.2029 & 0.1123 \\ 0.0608 & 0.6240 \end{matrix}$	**X**
Even Sampling w/ Standardized Data (Trick)	**X**	$\begin{matrix} 99.0896 & 0.2489 \\ 0.0707 & 0.5908 \end{matrix}$

Worst Case Image Results

RESULTS	***Naive-Bayes***	***Soft-Margin SVM***
Random Sampling		
Even Sampling		
Random Sampling w/ Cross-Validation		**X**
Even Sampling w/ Standardized Data (Trick)	**X**	

Best Case Image Results

RESULTS	***Naive-Bayes***	***Soft-Margin SVM***
Random Sampling		
Even Sampling		
Random Sampling w/ Cross-Validation		**X**
Even Sampling w/ Standardized Data (Trick)	**X**	

- ◆ Confusion Matrix: Top Left - True Negative; Top Right - False Positive
  Bottom Left - False Neative; Bottom Left - True Positive
- ◆ Best Case Confusion Matrix Image Performance:
  - Highest 'True Negative' + 'True Positive Score'
- ◆ Worst Case Confusion Matrix Image Performance:
  - Lowest 'True Negative' + 'True Positive Score'
- ◆ Average Case Confusion Matrix Image Performance:
  - The average percentage score of all 14 test images.
- ◆ Cross-Validation - Kfolds = 10
- ◆ Standardized Data Trick:
  - All features were standardized.
  - The standardized mapping was saved so that it could be applied to the test data.
  - For Features X & Y, both the training data and testing data were squared so that the center values would be close to 0 while everything further away from 0 would be above 0. This helps make our data slightly more linearly separable.

Observations:

- Random sampling greatly improved the performance of the Naive-Bayes Model over even sampling, while the opposite was true for our Soft-Margin SVM Model.
- The Soft-Margin SVM with Even Sampling and the Standardization Trick had the capacity to perform better than the regular Soft-Margin SVM with Even Sampling (see Best Case Confusion Matrix), while the regular Soft-Margin SVM with Even Sampling did better on average and in its worst case.
- Having a higher combined score for True Positive and True Negative isn't necessarily all that useful. For example, the Soft-Margin SVM with Random Sampling scored fairly well, but it did not even try to predict where the hole was in certain instances (0 True Positive score).

Moving Forward

Our next step is to see if we can expand the success of Project 2 to train another Naive-Bayes model to identify the coral instances that are highlighted in blue on the same set of manually labeled images. What we lose moving from the hole classification problem to the coral classification problem is locational consistency. The highlighted coral are scattered randomly in each training image, rendering the coordinate features useless. The color features may not be enough to carry the weight of distinguishing the coral instances as classifiable. One way we might combat this loss is by reimagining the noise removal portion of our image processing to favor a range of clusters that is typical of these coral instances. Additionally, we will experiment with pre-existing image identification neural networks and see if there is any potential for using one to recognize any one (or more) of the taxonomic groups. We hope to find more useful feature extraction tools from the 2D Image Recognition Layer (i.e. Histogram of Orientated Gradients) that will make isolating corals (and possibly other taxonomic groups) easier.