AN EXPLORATION OF MACHINE LEARNING TECHNIQUES FOR TAXONOMY

MOLLY BAIRD AND ASTRID LILJENBERG

Abstract. In this report we create a machine learning algorithm for classifying marine mammal vocalizations. The algorithm is based on building spectrograms of the vocalizations and utilizing the singular value decomposition to transform the data into the optimal basis for analysis. The results from using different unsupervised and supervised classifiers in the algorithm are compared. Sound clips from a number of different species of dolphins, whales, and pinnipeds (seal-like mammals) are used to evaluate the algorithm.

1. Introduction and Overview

Taxonomy, the science of classifying organisms, is vital to understanding the evolution of the world in which we live. Classically, biologists decide how plants and animals are related using their own inference– but what if we could create an algorithm that classified animals with little (or even without *any*) prior subjective knowledge? Equipped with time frequency analysis, machine learning, and data from the Watkins Marine Mammal Sound Database[1], this is what we attempt to achieve.

We focus on the case of marine mammals. We use twenty-five vocalization sound clips each from nine animals in three categories: pinnipeds (walrus, harp seal, ross seal), dolphins (atlantic spotted dolphin, white sided dolphin, and killer whale), and whales (humpback whale, sperm whale, and white beluga whale). We test whether our supervised machine learning algorithms classify each recording accurately, both within a category of animal (can it distinguish between types of whales, e.g.) and whether the algorithms can classify the animals across all categories (can it decide if a sound clip belongs to a pinniped, and then specifically which species of pinniped, e.g.).

2. Theoretical Background

2.1. **Time Frequency Analysis.** The main tool for analyzing frequency content is the Fourier transform, which decomposes a function into its respective frequencies. The absolute value of the Fourier transform represents the amount of each frequency that is present in the function. The Fourier transform and the inverse Fourier

Date: March 22, 2019.

[1]https://cis.whoi.edu/science/B/whalesounds/index.cfm

transform are given by

$$F(k) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{-ikx} f(x) dx, \tag{1}$$

$$f(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{ikx} F(k) dk, \tag{2}$$

where the k's are the wavenumbers.

The main drawback of using the Fourier transform is that the time information in the signal is lost, i.e. there is no way of knowing when in time a specific frequency was produced. For non-stationary signals, this is a problem. The Gábor transform aims to solve this problem by providing information in both time and frequency. The Gábor transform of a function f is defined as

$$\mathcal{G}[f](t, \omega) = \int_{-\infty}^{\infty} f(\tau) \bar{g}(\tau - t) e^{-i\omega\tau} d\tau \tag{3}$$

where $g(\tau - t)$ is some filter (Gábor window), and $\bar{g}$ its complex conjugate. One can think of this as the filter g sliding across the time signal, resulting in only a filtered part of the function f being Fourier transformed at each instant. The frequency content can thus be traced back to a specific time window in the signal.

When using the Gábor transform numerically, one chooses a filter, and lets the filter "slide" over the signal using a finite number of translations of the filter. For each location of the filter, the Fast Fourier Transform is applied to get the frequency content in this signal window. The frequency content of the entire signal can then be plotted as a spectrogram, where it is visualized as a function of both time and frequencies.

A basic Gaussian filter is given by

$$F(t) = e^{-a(t-\tau)^2}, \tag{4}$$

where the parameter a determines the width of the Gaussian and τ is the translation in t.

One important aspect to keep in mind is that increased time resolution, i.e. using a narrower Gábor window when filtering, results in decreased frequency resolution, and vice versa. When using narrow time windows, low frequencies will not be picked up as their wavelengths are longer than the window width. When using wide windows, lower frequencies will be captured, but, since the time window is wide, it is impossible to tell exactly when in time the frequencies originated.

2.2. **The Singular Value Decomposition (SVD).** Any matrix $M \in \mathbb{C}^{m \times n}$ represents a linear transformation that can be characterized by the manner in which it maps the unit n-ball.

Under this mapping by M, the unit ball is stretched into a hyper-ellipse, which can be identified by the lengths of its principal semi-axes. These lengths $\{\sigma_1, \ldots, \sigma_r\}$ are what we call the *singular values* of the matrix M. Note that this geometric definition as a length will mean that all the singular values are taken to be positive.

The singular value decomposition (SVD) of a matrix M is a factorization of the form $M = U\Sigma V^*$ such that $\text{diag}(\Sigma) = [\sigma_1, \ldots, \sigma_n]$, and $U \in \mathbb{C}^{m \times m}$ and $V \in \mathbb{C}^{n \times n}$ are unitary matrices (matrices that preserve length of vectors) whose columns are the left and right singular vectors of M (respectively).

Worth noting are the following facts about singular values and the SVD:

- *Every* matrix has an SVD.
- Singular values are uniquely determined up to complex scalars of norm 1.
- $r = rank(M)$ =(the number of nonzero singular values of M).
- It is standard to arrange the singular values in decreasing order along the diagonal so that $\Sigma_{1,1} = \sigma_1 \geq \sigma_2 \geq \cdots \sigma_r \geq 0$.
- Singular values of M are the square roots of the eigenvalues of M^*M where M^* is the conjugate transpose of M.
- The best *rank-k approximation* to a matrix $M = U\Sigma V^*$ is $\sum_{i=1}^{k} \sigma_i u_i v_i^*$.

The SVD is directly related to data driven modeling because of its relationship to the *covariance matrix*.

Given a matrix $M \in \mathbb{C}^{m\times n}$ of data measurements, its covariance matrix is $C_M = \frac{1}{n-1} MM^T$. This is a symmetric matrix whose diagonal entries represent the variance of the measurements in M, and whose off diagonal entries represent the covariances between measurement types.

The covariance matrix allows one to gain insight into redundancy patterns in the data contained in M. Removal of redundancy in data is a crucial step in data analysis, because redundancy suggests statistical dependency between measurements. This dependency must be removed before drawing any conclusions.

Diagonal entries of C_M which are larger relative to other diagonal entries correspond to dynamics of interest. Off-diagonal entries of the covariance matrix reveal the degree of redundancy of two measurements: larger off diagonal entry means more redundancy, small off diagonal entry means less redundancy and more statistical independence.

Diagonalizing (finding the eigendecomposition of) the covariance matrix, and then ordering these diagonal entries by magnitude reveals the principal components of the covariance matrix. This allows one to do away with the dynamics that are not of interest while also reducing the redundancy. (Note: the eigendecomposition will exist since M^TM is Hermitian positive semi-definite).

Since the squares of the singular values of M are the eigenvalues of the matrix M^TM, the SVD and the eigen-decomposition of the covariance matrix go hand in hand in determining the basis in which our data "wants" to be displayed.

For an $n \times n$ singular value matrix Σ, the energy of the modes, i.e. the percentage of information captured by the first k modes is

$$\frac{\sum_{i=1}^{k} \sigma_i}{\sum_{i=1}^{n} \sigma_i} \tag{5}$$

2.3. **Machine Learning.** Machine learning algorithms are a broad set of optimization techniques that allow one to model data with few starting assumptions. The general idea of machine learning is that the scientist gives the algorithm some data, and the algorithm then "learns" or reveals the governing structures within the data. The algorithms vary based upon how much data one has, how noisy that data is, and how much the scientist knows about their data before implementation of the algorithm.

There are two overarching types of machine learning algorithms: unsupervised learning and supervised learning. In both cases, one begins with a set $\mathcal{D}$ of data (that is presumably a subset of some theoretical set of all possible elements of that data, i.e. $\mathcal{D}$ =20 dogs is contained in the set of all dogs on Earth). This data is the input to the algorithm, and the algorithm outputs a set L of labels, one label for

each element of $\mathcal{D}$. Perhaps there are only two labels, meaning that all the data is in one of two categories, or perhaps there are many. This is is the framework of unsupervised learning: the scientist gives the algorithm some data, and the algorithm labels it. With supervised learning, the input is not only the set $\mathcal{D}$, but also a set Y of labels for the algorithm to choose from. Thus in the supervised case the scientist is effectively imparting more information onto the model by saying, "I know that my data should be classified in these exact labels."

We now outline the theory behind the machine learning algorithms employed in this project:

2.3.1. *k-means.* A simple unsupervised classification algorithm is the k-means method. The name refers to the goal of finding k cluster centers such that their means μ_j satisfy

$$\text{argmin}_{\mu_j} \sum_{j=1}^{k} \sum_{x_j \in \mathcal{D}'_j} ||x_j - \mu_j||^2 \tag{6}$$

where the x_j's are the data points and $\mathcal{D}'_j$ is the subdomain associated with cluster j. This optimization problem is solved through heuristic algorithms, such as Lloyd's algorithm, because it is a NP hard problem. One important property is that there is not necessarily only one unique output from the algorithm, but it depends on the initial guess of the cluster centers. If the centers are randomized at the beginning, this means the algorithm could potentially produce different results each time. Once the cluster centers have been decided, new data points are labeled based on the closest cluster center. Just like other unsupervised classifiers, the k-means method is generally not as good as supervised algorithms.

2.3.2. *Hierarchical Clustering with the Dendrogram.* Another unsupervised machine learning algorithm for clustering (and for visualizing those clusters) is the dendrogram. A dendrogram is a discrete tree whose leaves are the data points and whose branches represent the closeness of those data points to one another.

Hierarchical clustering methods generally take one of two approaches: agglomerative (bottom up) or divisive (top down). In an agglomerative approach, each data point begins as its own singleton cluster, and the data is re-clustered until every data point finally ends up in one large cluster. A divisive approach is the reverse scenario where all the data begins in one cluster and is then split up until finally every data point is in its own cluster.

The dendrogram algorithm we focus on here proceeds via an agglomerative method that makes decisions based on distance between points. "Distance" is the (vector) norm of the difference between the centers of two clusters. The choice of vector norm (euclidean, square euclidean, one norm, infinity norm, etc.) greatly influences the patterns in the dendrogram.

The algorithm itself proceeds in the following steps:

- The distance between all n data points x_j is computed.
- The closest two data points are merged into a single data point equidistant from these two data points (the center of a new cluster).
- Repeat the first two steps with the new $n-1$ data points.
- Stop when $n = 1$.

A threshold is then chosen which dictates the labeling of where each point belongs in the hierarchical scheme. Varying the threshold value gives different numbers of clusters.

2.3.3. *Support Vector Machines.* Support vector machines (SVM) were first developed in 1963 by Vapnik and Chervonenkis. The basic idea of SVM is to construct a function which separates data into clusters. The cluster on one side of the separation function gets one label, and the cluster on the other side gets a different label.

We classify SVM in terms of the type of its separation function: linear and nonlinear. The goal of linear SVM is to construct a hyperplane $H(x) = w \cdot x + b = 0$ which separates the data and also optimizes the largest margin between the data. We formulate this optimization problem via a loss function l

$$l(y_j, \overline{y}_j) = l(y_j, \text{sign}(w \cdot x_j + b)) = \begin{cases} 0 & \text{correctly labeled data} \\ 1 & \text{incorrectly labeled data} \end{cases} \tag{7}$$

This way, each mislabeled point increases the value of the penalty (loss) function. Minimizing this function obtains the optimal line, and our optimization problem becomes

$$\text{argmin}_{w,b} \sum_{j=1}^{m} l(y_j \overline{y}_j) + \frac{1}{2}||w||^2 \quad s.t. \quad \min_j |x_j \cdot w| = 1 \tag{8}$$

Standard optimization techniques like gradient descent (to get to the extrema) are then carried out to find the optimal value.

The linear SVM is then extended to the nonlinear case in which our hyperplane is composed with some nonlinear function of x:

$$H(\Phi(x)) = w \cdot \Phi(x) + b = 0 \tag{9}$$

In this case, Φ may operate on the coordinates of x by way of polynomials, trigonometric functions, a combination of the two, or something else. Optimization techniques for minimzation then become more complicated fairly quickly, as gradient descent may not return the absolute min of a nonlinear (and thus potentially nonconvex) function.

One issue with this method that arises with large data sets is the so-called "curse of dimensionality," where the optimization problem in the SVM method becomes computationally intractable. One way to fix this is via kernel methods, which essentially allows us to represent Taylor series expansions of a large number of observables in a compact way. The kernel function enables one to operate in a highdimensional, implicit feature space without ever computing the coordinates of the data in that space, but rather by simply computing the inner products between all pairs of data in the feature space.

2.3.4. *Naive Bayes.* The naive Bayes classifier is based on the famous Bayes' theorem for conditional probability, and is thus a probabilistic classifier. For a new data point $\mathbf{x}$, the probability of the data point belonging in a class C_k can be expressed

by Bayes' theorem as

$$\mathbb{P}(C_k|\mathbf{x}) = \frac{\mathbb{P}(C_k)\mathbb{P}(\mathbf{x}|C_k)}{\mathbb{P}(\mathbf{x})}. \tag{10}$$

Class labels $\hat{y} = C_k$ are assigned by solving

$$\hat{y} = \text{argmax}_k \mathbb{P}(C_k) \prod_{i=1}^{n} \mathbb{P}(x_i|C_k), \tag{11}$$

where $\mathbf{x} = (x_1, \ldots, x_n)$. A naive Bayes model can be constructed using the built-in function `fitcnb` in MATLAB, and can then be used to predict the labels of new data points.

2.3.5. *Cross Validation.* A crucial final step for all methods is cross-validation. This tests the algorithm with data that is already labeled (if labels are available). One gives the algorithm the data, sees how the algorithm labels it, and then compares these labels with the true labels. This is important for determining the accuracy of the algorithm, and for potentially editing the methods if the algorithm is inaccurate.

3. **Algorithm Implementation and Development**

The algorithm is structured into five main parts: readying the data, the k-means algorithm, the dendrogram, the support vector machine, and naive bayes.

3.1. **Readying the Data.** First, a for loop reads each of the 225 sound clips (9 animals, 25 samples each) into MATLAB and saves their sample rate and total time length of the clip.

All clips are trimmed to be the minimum time length of all clips. Then, using the sample rate, the clips are resampled to the same sample rate so that each data vector has the same dimension and corresponds to sound clips of the same time length.

A spectrogram of each normalized sound clip is computed using the `spectrogram` function in MATLAB. A Gaussian window (`gausswin` in MATLAB) is used for the spectrograms. Different window widths were compared, but the initial window width of 100 points was chosen by inspecting the localization of a sample spectrogram. Every spectrogram is then reshaped into a column vector and these columns are collected into a matrix X.

X is now a matrix with 9×25 columns, with each 25 column chunk corresponding to the samples for one species of animal. The absolute value of X is taken (as we're interested in the amount of a frequency at a certain time), and the order of the columns within each 25-column chunk are randomized. This ensures that the results of our supervised algorithms are not too dependent on which samples are used as training data and which are used as testing data.

The mean of each column of the randomized matrix is subtracted from itself to center the data around 0, and then the singular value decomposition is computed using the `svd` command in MATLAB.

3.2. **k-means.** For k-means estimation, the `kmeans` function in MATLAB is used directly on the X matrix with a k value of 9. This value was chosen because we have the prior knowledge that there are 9 species of animals to classify.

3.3. **Dendrogram.** The Euclidean distance was chosen for the implementation of the dendrogram algorithm. The algorithm itself is carried out and visualized by the `dendrogram` command in MATLAB, and it is applied directly on the X matrix. Various thresholds are experimented with, and ultimately $0.85 * m$ where m is the max of the third column of the matrix generated by the `linkage` function is chosen.

A dendrogram is created to classify all 225 sound clips. In addition, a bar graph is plotted with the weights of each data point within the dendrogram. Data points corresponding to the same species would ideally have the same weight in the dendrogram.

3.4. **Support Vector Machine.** After deciding which modes to include (based on the relative sizes of the singular values), the training data vector is created, and the labels are assigned to that data. The proportion of data used for training and testing is written as a variable with which we experiment. However, it is not the raw data that is used, but rather the first k columns of the matrix V from the SVD that corresponded to the k modes of importance. Then the vector of the true labels of the test data for cross validation is created. The functions `fitcecoc, predict, crossval, kfoldloss` in MATLAB are then employed to run the SVM analysis and cross validation. Note that `fitcecoc` is used rather than `fitsvm` because there are more than 2 desired labels.

3.5. **Naive Bayes.** The algorithm using the naive Bayes classifyer is identical to that of the SVM described above, the only difference being that the MATLAB command is `fitcnb` instead of `fitcecoc`. The same number of modes are used, and the results are cross validated in the same way as above.

4. Computational Results

The shortest sound clip was of length 1.0111 seconds, and the smallest sample rate was 5120, so all clips were trimmed to that length and resampled to that sample rate.

4.1. **Spectrograms.** When computing the spectrograms of the sound clips, we used a Gaussian window of size 100. A sample spectrogram of one of the killer whale clips is illustrated in figure 1.

4.2. **Singular Values.** The first 100 singular values from the SVD are illustrated in figure 2. They correspond to almost 80% of the energy in the modes, so we decided to truncate the matrix V at rank 100.

4.3. **k-means.** The results from using the k-means method with $k = 9$ to classify the data in X is visualized as a histogram in figure 3, where we can see the number of clips in each of the clusters. We know a priori that there should be an equal amount of clips in each of the 9 clusters, but here almost two thirds are clustered together.

4.4. **Dendrogram.** The dendrogram gave four clusters, one of which was much larger than the rest (see figure 4). The clustering outcome for each sample was plotted as a bar graph (see figure 5), with red vertical lines dividing the samples that corresponded to the same species.

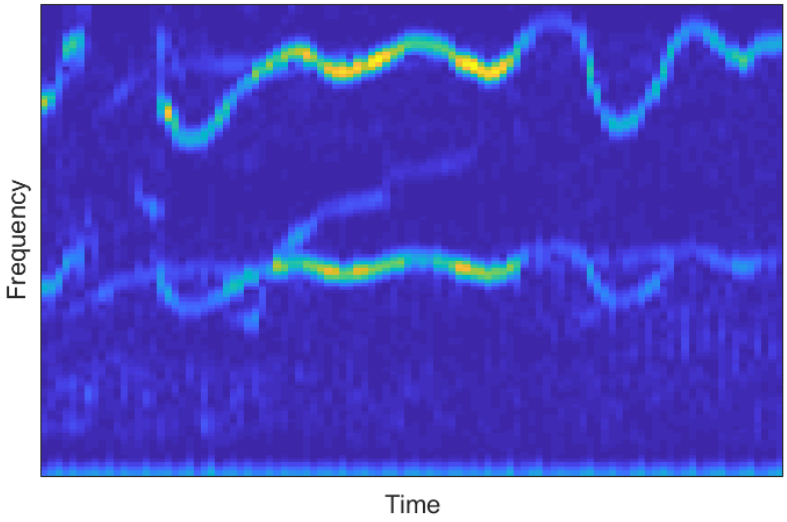

FIGURE 1. Spectrogram of a killer whale sound clip.

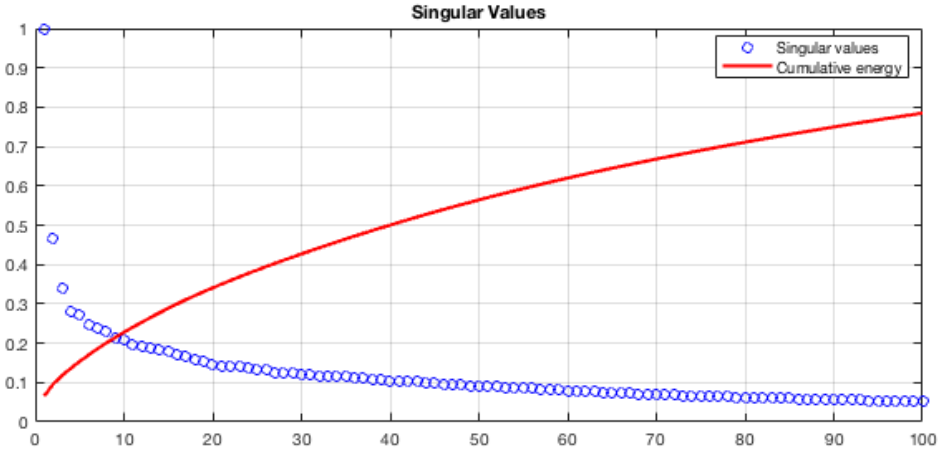

FIGURE 2. The first 100 (normalized) singular values plotted together with the cumulative energy in the modes.

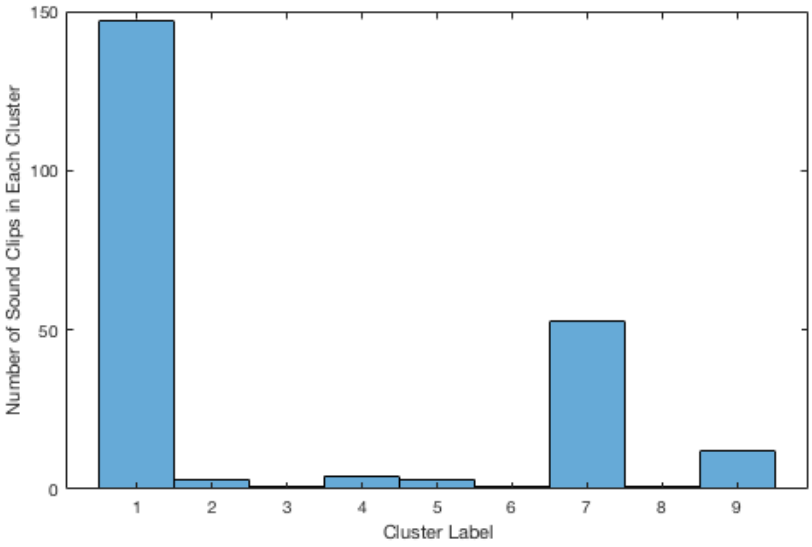

FIGURE 3. Histogram showing the number of sound clips grouped into each cluster by the k-means method.

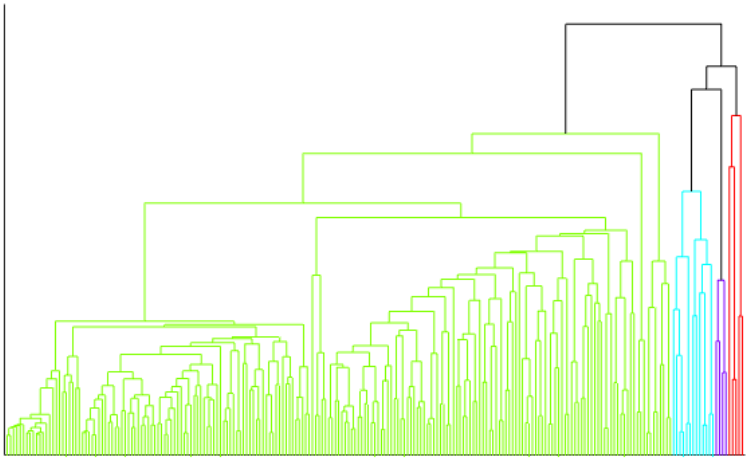

FIGURE 4. Dendrogram tree illustrating the evolution of the classification, with different colors corresponding to different clusters.

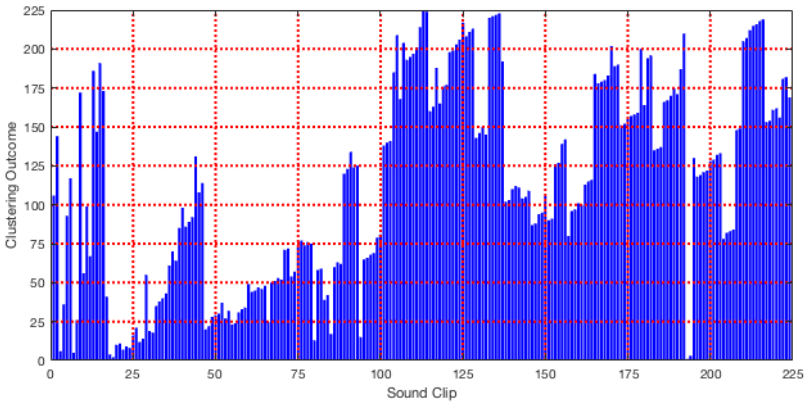

FIGURE 5. Clustering outcome of dendrogram routine. Ideally, each group of 25 along the Sound Clip axis would have a height within 25 of one another. The algorithm mislabeled many clips.

4.5. **Support Vector Machine.** We used 20 clips of each animal as a training set, and the remaining 5 as a test set. We ran two experiments with the SVM.

In the first experiment, we used all 9 species, and tested how accurately the algorithm could assign a sound clip to its specific species. The overall accuracy for this was 33%. The class loss was 0.7056. We also analyzed how well the algorithm categorized each sound clip into type of species (pinniped, whale, or dolphin). The success rates are presented in table 1, where the columns sum to 100% and the diagonal cells are the percentage of correct classifications.

TABLE 1

SVM	New Pinnipeds	New Dolphins	New Whales
Pinnipeds	33.33%	13.33%	20%
Dolphins	40%	46.67%	26.67%
Whales	26.67%	40%	53.33%

In the second experiment, we tested whether the algorithm could assign a sound clip to a particular species within one category (for example, can it label a dolphin sound clip specifically as a white sided dolphin). This second experiment was run three times, and in each run only the 3*25 sound clips corresponding to one category of animal were used in the training and test data. Within the pinnipeds, the class loss was 0.5167 and the overall accuracy was 0.5333. Within the dolphins, class loss was 0.3833 and overall accuracy was 0.5333. Within the whales the class loss was 0.4667 and the overall accuracy was 0.5333. (Note: it is curious that the overall success rate was 0.5333 for each of the categories. This was investigated thoroughly and is believed to be a coincidence rather than a coding error.)

4.6. **Naive Bayes.** We again used 20 clips of each animal as a training set, and the remaining 5 as a training set. The same two experiments as in the SVM section were run.

In the first experiment (with all 9 species), the overall accuracy when using

the naive Bayes algorithm was 40%. Again, we computed how well the algorithm classified the clips into the three overall categories. The results are presented in table 2. The diagonal cells show the percentage of correct classifications. The class loss was 0.71.

TABLE 2

Naive Bayes	New Pinnipeds	New Dolphins	New Whales
Pinnipeds	33.33%	0%	6.67%
Dolphins	20%	66.67%	26.67%
Whales	46.67%	33.33%	66.67%

The second experiment was carried out for naive bayes as well, and in each run of the experiment only the 3×25 sound clips corresponding to one category of animal were used in the training and test data. Within the pinnipeds, the class loss was .3833 and the overall accuracy was 0.5333. Within the dolphins the class loss was 0.4833 and the overall accuracy was 0.8. Within the whales the class loss was 0.4333 and the overall accuracy was 0.4667.

5. **Summary and Conclusions**

If one were to blindly guess what animal a random sound clip belonged to, the likelihood of success would be 11% (1/9). In the case of classifying all nine animals, our supervised learning algorithms reported success rates of 33% and 40%– three to four times as high as randomly guessing. Thus, we consider this portion of the experiment an overall success. In the case of classifying sound clips within one category (harp seal vs. ross seal vs. walrus, e.g.), blind guessing would yield a 33% (1/3) success rate. Our supervised algorithms had an average success rate of 56.6%, which is still better than guessing (although not three times better as in the previous case). This aligns with what we would expect from these classification algorithms; it should be more difficult to distinguish similar animals from one another than to distinguish very different animals from one another.

The unsupervised algorithms performed very poorly. The k-means method put many sound clips all in one cluster, while the other clusters were much smaller (see Figure 3). This performed so badly that we did not cross-validate the model, as it was clear from the histogram how poor the results were. The dendrogram gave a similarly poor result, putting most clips in one cluster, as seen by the green branches in Figure 4.

The first singular value was relatively much larger than all the others. Perhaps this explains some of the tendency to put many animals in just one category.

This experiment could be improved in many ways. It is likely that the best way the results could be improved would be if the sound clips were longer. We experimented with splicing the sound clips in the middle rather than the beginning, but results did not improve. A larger number of sound clips would also likely improve the results. Also, many of the clips were from the 1960's, so data with less noise recorded on newer equipment might improve results.

Overall, this project demonstrates the power of supervised learning over unsupervised learning. These methods could definitely be extended to ask questions related to any variety of organism, and could prove to be powerful tools in the science of taxonomy.

6. **Appendix A: MATLAB Functions used**

- `abs` Returns the absolute value
- `audioread` Reads in a .wav file
- `cell(M,N)` creates an M times N cell array
- `cd` changes the current directory
- `crossval` Runs cross validation
- `dendrogram, pdist, linkage` MATLAB's built in dendrogram functions
- `diag` Returns the diagonal of a given matrix as a vector
- `dir` Goes to the directory with a given name
- `double` Converts uint8 into doubles
- `fitcecoc` Runs the SVM algorithm but for any desired finite number of labels
- `fitcnb` returns a naive Bayes model given some data and their labels
- `flipud` Flips an image in the up-down direction
- `floor` rounds a decimal number down to the nearest integer
- `fullfile` Creates a URL to a given folder within a directory
- `gausswin` returns a gaussian window
- `histcounts(X,EDGES)` counts the number values in X between each of the edges
- `imagesc(X)` displays the image in X but rescaled so that the full colormap is used
- `kfoldLoss` Measures the accuracy of the SVM algorithm
- `kmeans(X,k)` labels the data in X using k labels
- `length(X)` returns the length of the vector X
- `mean` Returns the average
- `min(X) max(x)` returns the min/max in each column if X is a matrix, and the overall min/max if X is a vector
- `pcolor` Plots an images with a specific color scheme
- `predict` Predicts the label of the test data given the results of SVM
- `randperm(N,K)` returns a vector with K unique integers randomly selected from 1:N
- `resample(X,P,Q)` resamples the data in X at P/Q times the original sample rate
- `reshape` Turns our data into a matrix of a prescribed size
- `save` Saves an object with a designated name
- `size(X)` returns the size of a matrix X
- `spectrogram(X)` Creates a spectrogram of the data in X
- `svd, 'econ'` Returns three matrices, the Singular value decomposition of a desired matrix, 'econ' to handle a larger matrix
- `zeros(M,N)` returns a M times N matrix of zeros
- `plot`, `title`, `xlabel`, `ylabel`, `legend` and `set(gca,'xtick',[])` were used plot, and to set the titles, axis labels, and legends of plots, and to turn off the axes ticks

7. Appendix B: MATLAB Code

```
% LOAD AND TRIM THE DATA

nSpecies = 9;
nSamples = 25; % per species

species = {'Walrus','Harp_seal','Ross_seal',...
    'Killer_whale','Atlantic_spotted_dolphin','
       White_sided_dolphin',...
    'Humpback_whale','Sperm_whale','White_whale'};

data = cell(nSpecies,nSamples);
lengths = zeros(nSpecies,nSamples);
FSs = zeros(nSpecies,nSamples);

cd Final582DATA
for i = 1:nSpecies
    cd(species{i});
    files = dir('*.wav');
    for j = 1:nSamples
        [s,Fs] = audioread(files(j).name);
        l = length(s);
        slength = l/Fs;
        lengths(i,j) = slength;
        FSs(i,j) = Fs;
    end
    cd ..
end
cd ..

minLength = min(min(lengths));
minFs = min(min(FSs));

cd Final582DATA
for i = 1:nSpecies
    cd(species{i});
    files = dir('*.wav');
    for j = 1:nSamples
        [s,Fs] = audioread(files(j).name);
        s = s(1:floor(Fs*minLength));
        s = resample(s,minFs,Fs);
        data{i,j} = s;
    end
    cd ..
```

```
end
cd ..

%%
window = 100;
[H,W] = size(spectrogram(data{1,1},gausswin(window)));
X = zeros(H*W, nSpecies*nSamples);
counter = 0;

for i = 1:nSpecies
    for j = 1:nSamples
        counter = counter + 1;
        X(:,counter) = reshape(spectrogram(data{i,j},window
            ),[],1);
    end
end

% plotting a spectrogram
figure(1); clf
imagesc(flipud(abs(spectrogram(data{2,5},gausswin(window)))
    ))
set(gca,'xtick',[])
set(gca,'ytick',[])
xlabel('Time')
ylabel('Frequency')
%title('Spectrogram of One Sound Clip')


X = abs(X);



%%   randomize the columns of X, within each 25 column
    chunk
X_r = [];
for i = 1:9
    M_new = [];
    k = 25*(i-1)+1;
    M_curr = X(:,k:k+25-1);
    y = randperm(25,25);
    for j = 1:25
        l = y(j);
        M_new = [M_new M_curr(:,l)];
    end
    X_r = [X_r M_new];
```

```
85  end
86
87  %% Take the SVD
88  [U,S,V] = svd(X_r-mean(X_r),'econ');          %take the SVD
89  s = diag(S)/max(diag(S));
90
91  clf
92
93  figure(1)
94  plot(s(1:100), 'bo');
95  hold on
96  grid on
97  plot(cumsum(s(1:100))/sum(s), 'r', 'Linewidth',2)
98  title('Singular Values')
99  legend('Singular values','Cumulative energy', 'location', '
        best')
100
101
102 %% Set Up the training data, test data, and label matrices
103
104 features = 1:100;
105
106 n = 20;      %training
107 m = 5;       %testing
108 label = [];
109 xtrain = [];
110 test = [];
111 truth = [];
112
113 %training
114 for i = 1:9
115     label = [label; i*ones(n,1)];                %labels
116     k = 25*(i-1)+1;
117     xtrain = [xtrain; V(k:k+n-1,features)];       %training
118     l = n+k;
119     test = [test; V(l:l+m-1,features)];          %test data
120     truth = [truth; i*ones(m,1)];
121 end
122
123
124
125 %% %%%%%%%%%%%%%% SVM %%%%%%%%%%%%%%
126 Mdl = fitcecoc(xtrain,label);
127 test_labels = predict(Mdl,test);
128
```

```
129
130 CMdl = crossval(Mdl);
                                                % cross-validate
    the model
131 classLoss = kfoldLoss(CMdl)
                                            % compute class loss
132
133 overallAccuracy = sum(test_labels==truth)/length(truth)
134
135 T = zeros(9,9);
136 for i = 1:9
137     T(:,i) = histcounts(test_labels(m*(i-1)+1:m*i),0.5:9.5)
        ;
138 end
139 T = T./m*100;
140 disp(T);
141
142 T = zeros(3,3);
143 for i = 1:3
144     T(:,i) = histcounts(test_labels(3*m*(i-1)+1:3*m*i),[0.5
        3.5 6.5 9.5]);
145 end
146 T = T./(3*m)*100;
147 disp(T);
148
149
150
151 %% %%%%%%%%%%% DENDROGRAM %%%%%%%%%%%%%%%%%%
152
153 %%%% dendrogram for all 9
154 clf
155 Y = pdist(X_r,'euclidean');
156 Z = linkage(Y,'average');
157 thresh=0.001*max(Z(:,3));
158 [H,T,Outperm]=dendrogram(Z,225,'ColorThreshold',thresh);
159 set(gca,'xtick',[])
160 set(gca,'ytick',[])
161 %%
162 figure(2); clf
163 bar(Outperm,'b'), hold on
164 plot([0 225], [25 25], 'r:', [0 225], [50 50], 'r:', [0
    225], [75 75],'r:', [0 225], [100 100], 'r:',[0 225],
    [125 125],'r:', [0 225], [150 150], 'r:',[0 225], [175
    175], 'r:',[0 225], [200 200], 'r:','Linewidth',2)
```

```
165 plot([25 25], [0 225], 'r:', [50 50], [0 225], 'r:', [75
        75], [0 225],'r:', [100 100], [0 225], 'r:',[125 125],
        [0 225], 'r:',[150 150], [0 225], 'r:',[175 175], [0
        225],'r:', [200 200], [0 225], 'r:','Linewidth',2)
166 axis([0 225 0 225])
167 xlabel('Sound Clip')
168 ylabel('Clustering Outcome')
169 xticks(0:25:225)
170 yticks(0:25:225)
171
172
173
174
175 %% %%%%%%%%%%%%%% NB (naive bayes) %%%%%%%%%%%%%%
176 Mdl = fitcnb(xtrain,label);
177 test_labels = predict(Mdl,test);
178
179
180 CMdl = crossval(Mdl);
                                                % cross-validate
       the model
181 classLoss = kfoldLoss(CMdl)
                                             % compute class loss
182
183 overallAccuracy = sum(test_labels==truth)/length(truth)
184
185 T = zeros(9,9);
186 for i = 1:9
187     T(:,i) = histcounts(test_labels(m*(i-1)+1:m*i),0.5:9.5)
            ;
188 end
189 T = T./m*100;
190 disp(T);
191
192 T = zeros(3,3);
193 for i = 1:3
194     T(:,i) = histcounts(test_labels(3*m*(i-1)+1:3*m*i),[0.5
            3.5 6.5 9.5]);
195 end
196 T = T./(3*m)*100;
197 disp(T);
198
199
200
201
```

```
202
203
204 %% %%%%%%%%%% k-means %%%%%%%%%%%%
205 clf
206 k = 9;
207 labels = kmeans(X_r',k);
208 histogram(labels)
209 xlabel('Cluster Label')
210 ylabel('Number of Sound Clips in Each Cluster')
211
212
213
214 %% %%%%%%%%%%%% SVM FOR PINNIPEDS
215
216 features = 1:100;
217
218 n = 20;      %training
219 m = 5;       %testing
220 label = [];
221 xtrain = [];
222 test = [];
223 truth = [];
224
225 %training
226 for i = 1:3
227     label = [label; i*ones(n,1)];                %labels
228     k = 25*(i-1)+1;
229     xtrain = [xtrain; V(k:k+n-1,features)];       %training
230     l = n+k;
231     test = [test; V(l:l+m-1,features)];          %test data
232     truth = [truth; i*ones(m,1)];
233 end
234
235 Mdl = fitcecoc(xtrain,label);
236 test_labels = predict(Mdl,test);
237
238
239 CMdl = crossval(Mdl);
                                                % cross-validate
    the model
240 classLoss = kfoldLoss(CMdl)
                                          % compute class loss
241
242 overallAccuracy = sum(test_labels==truth)/length(truth)
243
```

```
244
245 %% %%%%%%%%%%%%% SVM FOR DOLPHINS
246
247 features = 1:100;
248
249 n = 20;      %training
250 m = 5;       %testing
251 label = [];
252 xtrain = [];
253 test = [];
254 truth = [];
255
256 %training
257 for i = 4:6
258     label = [label; i*ones(n,1)];                %labels
259     k = 25*(i-1)+1;
260     xtrain = [xtrain; V(k:k+n-1,features)];       %training
261     l = n+k;
262     test = [test; V(l:l+m-1,features)];          %test data
263     truth = [truth; i*ones(m,1)];
264 end
265
266 Mdl = fitcecoc(xtrain,label);
267 test_labels = predict(Mdl,test);
268
269
270 CMdl = crossval(Mdl);
                                                % cross-validate
    the model
271 classLoss = kfoldLoss(CMdl)
                                            % compute class loss
272
273 overallAccuracy = sum(test_labels==truth)/length(truth)
274
275
276 %% %%%%%%%%%%%%% SVM FOR WHALES
277
278 features = 1:100;
279
280 n = 20;      %training
281 m = 5;       %testing
282 label = [];
283 xtrain = [];
284 test = [];
285 truth = [];
```

```
286
287 %training
288 for i = 7:9
289     label = [label; i*ones(n,1)];                %labels
290     k = 25*(i-1)+1;
291     xtrain = [xtrain; V(k:k+n-1,features)];       %training
292     l = n+k;
293     test = [test; V(l:l+m-1,features)];          %test data
294     truth = [truth; i*ones(m,1)];
295 end
296
297 Mdl = fitcecoc(xtrain,label);
298 test_labels = predict(Mdl,test);
299
300
301 CMdl = crossval(Mdl);
                                                % cross-validate
    the model
302 classLoss = kfoldLoss(CMdl)
                                          % compute class loss
303
304 overallAccuracy = sum(test_labels==truth)/length(truth)
305
306
307
308 %% %%%%%%%%%%%% NB FOR PINNIPEDS
309
310 features = 1:100;
311
312 n = 20;      %training
313 m = 5;       %testing
314 label = [];
315 xtrain = [];
316 test = [];
317 truth = [];
318
319 %training
320 for i = 1:3
321     label = [label; i*ones(n,1)];                %labels
322     k = 25*(i-1)+1;
323     xtrain = [xtrain; V(k:k+n-1,features)];       %training
324     l = n+k;
325     test = [test; V(l:l+m-1,features)];          %test data
326     truth = [truth; i*ones(m,1)];
327 end
```

```
328
329 Mdl = fitcnb(xtrain,label);
330 test_labels = predict(Mdl,test);
331
332
333 CMdl = crossval(Mdl);
                                                    % cross-validate
    the model
334 classLoss = kfoldLoss(CMdl)
                                              % compute class loss
335
336 overallAccuracy = sum(test_labels==truth)/length(truth)
337
338
339
340 %% %%%%%%%%%%%% NB FOR DOLPHINS
341
342 features = 1:100;
343
344 n = 20;      %training
345 m = 5;       %testing
346 label = [];
347 xtrain = [];
348 test = [];
349 truth = [];
350
351 %training
352 for i = 4:6
353     label = [label; i*ones(n,1)];                %labels
354     k = 25*(i-1)+1;
355     xtrain = [xtrain; V(k:k+n-1,features)];       %training
356     l = n+k;
357     test = [test; V(l:l+m-1,features)];          %test data
358     truth = [truth; i*ones(m,1)];
359 end
360
361 Mdl = fitcnb(xtrain,label);
362 test_labels = predict(Mdl,test);
363
364
365 CMdl = crossval(Mdl);
                                                    % cross-validate
    the model
366 classLoss = kfoldLoss(CMdl)
                                              % compute class loss
```

```
367
368 overallAccuracy = sum(test_labels==truth)/length(truth)
369
370
371
372 %% %%%%%%%%%%%% NB FOR WHALES
373
374 features = 1:100;
375
376 n = 20;      %training
377 m = 5;       %testing
378 label = [];
379 xtrain = [];
380 test = [];
381 truth = [];
382
383 %training
384 for i = 7:9
385     label = [label; i*ones(n,1)];                %labels
386     k = 25*(i-1)+1;
387     xtrain = [xtrain; V(k:k+n-1,features)];       %training
388     l = n+k;
389     test = [test; V(l:l+m-1,features)];          %test data
390     truth = [truth; i*ones(m,1)];
391 end
392
393 Mdl = fitcnb(xtrain,label);
394 test_labels = predict(Mdl,test);
395
396
397 CMdl = crossval(Mdl);
                                                % cross-validate
    the model
398 classLoss = kfoldLoss(CMdl)
                                            % compute class loss
399
400 overallAccuracy = sum(test_labels==truth)/length(truth)
```