# Assignment 1: Raw Data to Feature Space CSC 410-01

CRISTIAN LORENZO-PAVON, UNCG, USA

The main focus of the Raw Data to Feature Space was to extract features from 3 different types of birds. It was also to learn what is Machine Learning (ML) and it's important aspect for big data classification, the challenges, techniques and methods, and the different things to look out.

## CHAPTER 1

The first chapter explains the field of data science and its progress on technological dilemmas and advancements. It focuses on two important paradigms, big data paradigm and machine learning paradigm. The important elements of data science are the data, the knowledge, and the operations. Data science can be interpreted as the analysis and extraction of useful information from managed data sets. While also having a clear understanding of the structure producing the data. A couple of real world applications that produce big data would be network intrusion detection systems and climate-change detection systems. The big data is hard to manage and analyze as it's extremely complex and unstructured. In order to have a good system you'd have to have a clear understanding of the terms, data and knowledge, and for the three operations, mathematical, logical, and physical.

A big issue in big data classification is the technology required to handle the volume, velocity, and variety of characteristics. It is necessary to have algorithms and mathematical models to efficiently classify the data. Regarding the technology dilemma, an issue is that there is no available technology that can efficiently analyze and extract useful information in an exponentially growing data set. There also is a challenge with the lack of approaches that can select an appropriate technique to solve the big data issue. If we invest into technology with the hope of solving those issues you can run the risk of finding out that the data obtained is not "big data " effectively ruining your ROI on the technology.

A technological advancement in big data classification would be the modern distributed file systems and the distributed machine learning. The technology named Hadoop is an example of distributed machine learning that processes big data. Two techniques which may help develop better learning techniques in the future would be decision tree learning and deep learning. The goal is to make accurate decisions and we do this by observing and learning the characteristics given the conditions.

## CHAPTER 2

Chapter 2 is more focused on the organization of big data to better contribute to the analytics of big data. This helps with presentations to be more simplified in their summary and of their objectives for an audience to more easily understand. There are three main components for big data which are analytics, classification, and scalability. They all have controllers that have different focuses. In the analytics the controllers work with data representation and extraction of the important information that will be presented. In big data classification, there are two main controllers that help with classification which would be the machine learning processes and the classification algorithms. The main focus for big data classification is the process of classifying the data under the conditions placed by the controller. In big data, scalability is always an unavoidable issue. To solve the scalability issue it's very important to extract low-dimensional structures from high-dimensional systems.

There are many business benefits that come from big data technologies and techniques as it can provide information about its current or future market. The three important controllers of big data would be class, feature, and observation characteristics. The class controller represents the variety and different types of events to help the big data to be better grouped. This controller also is the main cause for unpredictability in big data, making it more difficult to create ML models and algorithms. The feature controller defines the dimensionality by the scalability and parameters of the data. It is one of the major contributors for scalability issues in the big data paradigm. Since there is a massive result of volume and velocity the observation controller helps with the classification issues. It also manages, processes, and analyzes the data and has more difficulty with the growth of volume in data.

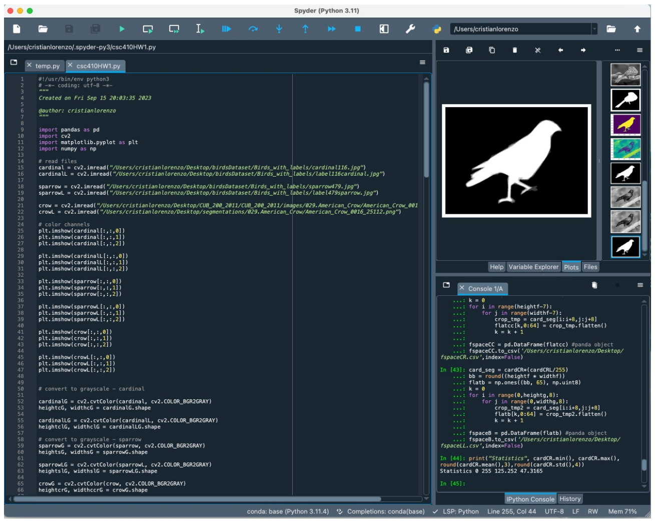
In big data there are many challenges but it can be broken into two categories: technique and technology. For techniques, the challenges are the classification, scalability, and analysis. The challenges regarding technology are computation, communication and storage. Problems that can arise from the observation controller would be the data processing, storage requirements, and communications issues. While the class controllers may have an impact on the classification techniques causing a decline in performance. A reduction of dimensionality or storage and computing power can impose a stress on the feature controller.

## CHAPTER 3

For chapter 3, its focus was the importance of the changes a set of data goes through when transforming into a set of big data. Having a more in-depth analysis of data can have more properties which can be used to anticipate the characteristics of the data in the future. The evolution of patterns is a common thing in big data environments. This is why current information may not be valid because you wouldn't know if the data has gone through evolution or not. If a data set has gone through evolution it would increase the complexity of the data making the mathematical models and algorithms more difficult for big data classification.

To understand the evolution of patterns a split-merge-split framework would help. A model that is great for big data analytics simulation would be the data expansion. This model can provide the deformation of patterns to better study the characteristics in the data. As it uses statistical and mathematical approaches it is based on the standardization, normalization, linear transformation, and orthogonalization.

## ENVIRONMENT



```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Fri Sep 15 20:03:35 2023

@author: cristianlorenzo
"""

import pandas as pd
import cv2
import matplotlib.pyplot as plt
import numpy as np

# read files
cardinal = cv2.imread("/Users/cristianlorenzo/Desktop/birdsDataset/Birds_with_labels/cardinal116.jpg")
cardinalL = cv2.imread("/Users/cristianlorenzo/Desktop/birdsDataset/Birds_with_labels/label116cardinal.jpg")

sparrow = cv2.imread("/Users/cristianlorenzo/Desktop/birdsDataset/Birds_with_labels/sparrow479.jpg")
sparrowL = cv2.imread("/Users/cristianlorenzo/Desktop/birdsDataset/Birds_with_labels/label479sparrow.jpg")

crow = cv2.imread("/Users/cristianlorenzo/Desktop/CUB_200_2011/CUB_200_2011/images/029.American_Crow/American_Crow_001
crowL = cv2.imread("/Users/cristianlorenzo/Desktop/segmentations/029.American_Crow/American_Crow_0016_25112.png")

# color channels
plt.imshow(cardinal[:,:,0])
plt.imshow(cardinal[:,:,1])
plt.imshow(cardinal[:,:,2])

plt.imshow(cardinalL[:,:,0])
plt.imshow(cardinalL[:,:,1])
plt.imshow(cardinalL[:,:,2])

plt.imshow(sparrow[:,:,0])
plt.imshow(sparrow[:,:,1])
plt.imshow(sparrow[:,:,2])

plt.imshow(sparrowL[:,:,0])
plt.imshow(sparrowL[:,:,1])
plt.imshow(sparrowL[:,:,2])

plt.imshow(crow[:,:,0])
plt.imshow(crow[:,:,1])
plt.imshow(crow[:,:,2])

plt.imshow(crowL[:,:,0])
plt.imshow(crowL[:,:,1])
plt.imshow(crowL[:,:,2])


# convert to grayscale - cardinal

cardinalG = cv2.cvtColor(cardinal, cv2.COLOR_BGR2GRAY)
heightcG, widthcG = cardinalG.shape

cardinalLG = cv2.cvtColor(cardinalL, cv2.COLOR_BGR2GRAY)
heightclG, widthclG = cardinalLG.shape

# convert to grayscale - sparrow
sparrowG = cv2.cvtColor(sparrow, cv2.COLOR_BGR2GRAY)
heightsG, widthsG = sparrowG.shape

sparrowLG = cv2.cvtColor(sparrowL, cv2.COLOR_BGR2GRAY)
heightslG, widthslG = sparrowLG.shape

crowG = cv2.cvtColor(crow, cv2.COLOR_BGR2GRAY)
heightcrG, widthccrG = crowG.shape
```

```
    ...: k = 0
    ...: for i in range(heightf-7):
    ...:     for j in range(widthf-7):
    ...:         crop_tmp = card_seg[i:i+8,j:j+8]
    ...:         flatcc[k,0:64] = crop_tmp.flatten()
    ...:         k = k + 1
    ...:
    ...: fspaceCC = pd.DataFrame(flatcc) #panda object
    ...: fspaceCC.to_csv('/Users/cristianlorenzo/Desktop/
fspaceCR.csv',index=False)

In [43]: card_seg = cardCR*(cardCRL/255)
    ...: bb = round((heightf * widthf))
    ...: flatb = np.ones((bb, 65), np.uint8)
    ...: k = 0
    ...: for i in range(0,heightg,8):
    ...:     for j in range(0,widthg,8):
    ...:         crop_tmp2 = card_seg[i:i+8,j:j+8]
    ...:         flatb[k,0:64] = crop_tmp2.flatten()
    ...:         k = k + 1
    ...:
    ...: fspaceB = pd.DataFrame(flatb) #panda object
    ...: fspaceB.to_csv('/Users/cristianlorenzo/Desktop/
fspaceLL.csv',index=False)

In [44]: print("Statistics", cardCR.min(), cardCR.max(),
round(cardCR.mean(),3),round(cardCR.std(),4))
Statistics 0 255 125.252 47.3165

In [45]:
```

Fig. 1. Spyder environment

The version of Python used is 'Python 3.11.4'. Running on Spyder from the Anaconda Navigator.

## IMAGE0

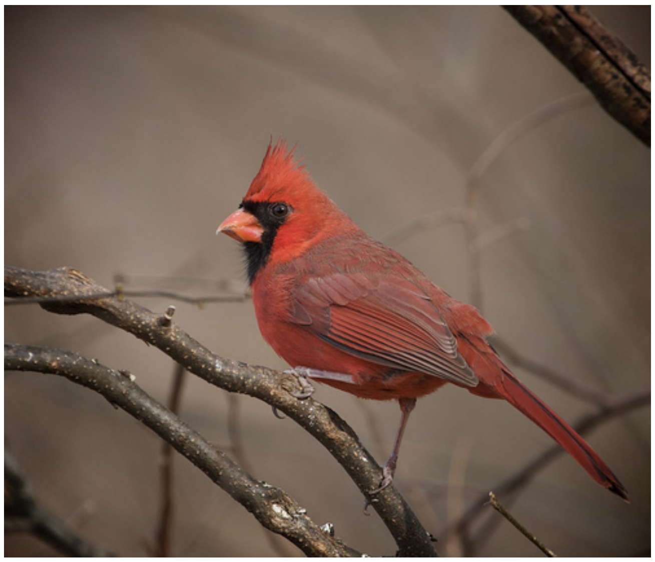


Fig. 2. Cardinal

I got the image0 from the Caltech Data library, folder CUB_200_2011. The reasoning behind choosing these images is because it was easy to download and has a whole data set of 200 different birds that can be used for research purposes. It was the third image from the cardinal folder. I chose this cardinal as it could provide many features. A close shot of the bird and facing the camera. The background is also blurred.

## IMAGE1

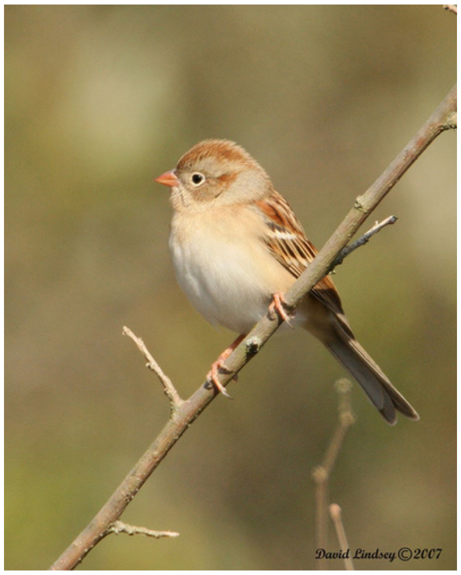


Fig. 3. Sparrow

I got the image1 from the Caltech Data library, folder CUB_200_2011. The reasoning behind choosing these images is because it was easy to download and has a whole data set of 200 different birds that can be used for research purposes. It was the fifth image from the sparrow folder. It is facing the same way and is a close up of the sparrow with the background blurred.

## IMAGE2

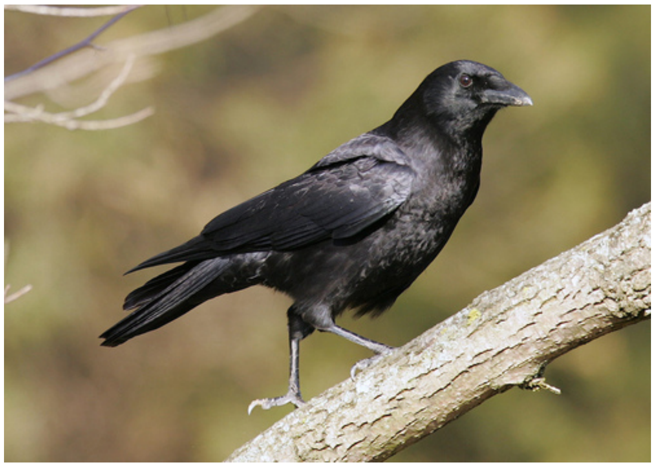
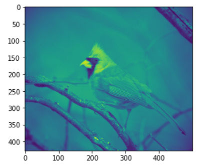


Fig. 4. Crow

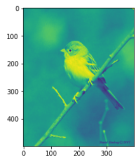
I got the image2 from the Caltech Data library, folder CUB_200_2011. The reasoning behind choosing these images is because it can be used for research purposes. It is different from the previous birds as it is black and lacks color.

## RGB CHANNELS

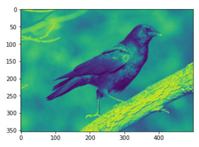
I then read the image files from my computer and display the RGB chaneels for all three birds: Cardinal, Sparrow, and Crow.



(a) Cardinal RGB Channel



(b) Sparrow RGB Channel

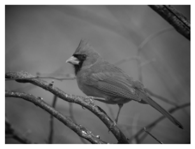


(c) Crow RGB Channel

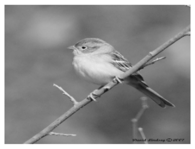
Fig. 5. RGB channels for all bird images

## BIRDS GRAY SCALE AND RESIZED

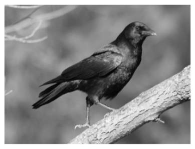
I then convert all bird images into grayscale and resize them to '240 x 320'.



(a) Cardinal Gray scale



(b) Sparrow Gray scale

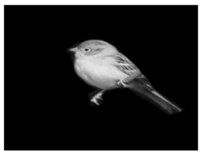


(c) Crow Gray scale

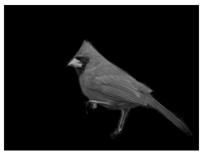
Fig. 6. Results from grayscale images of birds

## BIRD SEGMENTATION'S

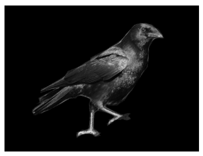
Then I save the cardinal, sparrow, and crow grayscale images with the segmentation's of the bird images.



(a) Sparrow Segmentation



(b) Cardinal Segmentation



(c) Crow Segmentation

Fig. 7. All three birds segmentation's

## BLOCK FEATURES

### Code implementation

```
# Overlapping for cardinal:
card_seg = cardC * (cardCL/255)
cc = round((heightc)*(widthc))
flatcc = np.zeros((cc, 256), np.uint16)
k = 0
for i in range(heightc-15):
    for j in range(widthc-15):
        crop_tmp = card_seg[i:i+16,j:j+16]
        flatcc[k,0:256] = crop_tmp.flatten()
        k = k + 1

fspaceCC = pd.DataFrame(flatcc) #panda object
fspaceCC.to_csv('/Users/cristianlorenzo/Desktop/fspaceI0.
csv',index=False)
```

**REVISED** This block of code creates block features of the cardinal bird in '16x16'. It uses the overlapping method to create the block features. After taking the block features it flattens the block and provides the feature space into a csv file.

```
# Overlapping for sparrow:
cc = round(((heightd)*(widthd)))
flatcc = np.zeros((cc, 256), np.uint16)
k = 0
for i in range(heightd-15):
    for j in range(widthd-15):
        crop_tmp = card_seg[i:i+16,j:j+16]
        flatcc[k,0:256] = crop_tmp.flatten()
        k = k + 1

fspaceSS = pd.DataFrame(flatcc) #panda object
fspaceSS.to_csv('/Users/cristianlorenzo/Desktop/fspaceI1.
csv',index=False)
```

This block of code creates block features of the sparrow bird in '16x16'. It uses the overlapping method to create the block features. After taking the block features it flattens the block and provides the feature space into a csv file.

```
#Overlapping for crow:
card_seg = cardCR * (cardCRL/255)
cc = round((heightf)*(widthf))
flatcc = np.zeros((cc, 256), np.uint16)
k = 0
for i in range(heightf-15):
    for j in range(widthf-15):
        crop_tmp = card_seg[i:i+16,j:j+16]
        flatcc[k,0:256] = crop_tmp.flatten()
        k = k + 1

fspaceCR = pd.DataFrame(flatcc) #panda object
fspaceCR.to_csv('/Users/cristianlorenzo/Desktop/fspaceI2.
csv',index=False)
```

This block of code creates block features of the crow bird in '16x16'. It uses the overlapping method to create the block features. After taking the block features it flattens the block and provides the feature space into a csv file.

## SLIDING BLOCK FEATURES

### Code Implementation

```
# Non-overlapping for cardinal:
card_seg = cardC*(cardCL/255)
bb = round((heightc * widthc))
flatb = np.ones((bb, 65), np.uint16)
k = 0
for i in range(0,heightb,8):
    for j in range(0,widthb,8):
        crop_tmp2 = card_seg[i:i+8,j:j+8]
        flatb[k,0:64] = crop_tmp2.flatten()
        k = k + 1

fspaceB = pd.DataFrame(flatb) #panda object
fspaceB.to_csv('/Users/cristianlorenzo/Desktop/
fspaceNN.csv',index=False)
```

After completing the overlapping block features, I continued to get the feature space using the sliding block method. In the snippet above, the cardinal image's feature space is turned into a CSV file unto my desktop.

```
# Non-overlapping for sparrow:
card_seg = cardS*(cardSL/255)
bb = round(((heighte)*(widthe)))
flatb = np.ones((bb, 65), np.uint16)
k = 0
for i in range(0,heighte,8):
    for j in range(0,widthe,8):
        crop_tmp2 = card_seg[i:i+8,j:j+8]
        flatb[k,0:64] = crop_tmp2.flatten()
        k = k + 1

fspaceB = pd.DataFrame(flatb) #panda object
fspaceB.to_csv('/Users/cristianlorenzo/Desktop/
fspaceMM.csv',index=False)
```

After completing the cardinals sliding block features, I continued with the sparrow's feature space using sliding block method. In the snippet above, the sparrow image's feature space is turned into a CSV file unto my desktop.

```
# Non-overlapping for crow:
card_seg = cardCR*(cardCRL/255)
bb = round((heightf * widthf))
flatb = np.ones((bb, 65), np.uint16)
k = 0
for i in range(0,heightg,8):
    for j in range(0,widthg,8):
        crop_tmp2 = card_seg[i:i+8,j:j+8]
        flatb[k,0:64] = crop_tmp2.flatten()
        k = k + 1

fspaceB = pd.DataFrame(flatb) #panda object
fspaceB.to_csv('/Users/cristianlorenzo/Desktop/
fspaceLL.csv',index=False)
```

After completing the overlapping block features, I continued to get the feature space using the sliding block method. In the snippet above, the crow image's feature space is turned into a CSV file unto my desktop.

## STATISTICAL DESCRIPTORS

I then print out some statistical facts from each bird such as their min, max, mean, and standard deviation. I get the following:

### Output

| bird | min | max | mean | standard deviation |
|---|---|---|---|---|
| Cardinal | 0 | 255 | 95.971 | 28.1819 |
| Sparrow | 3 | 253 | 125.619 | 29.0046 |
| Crow | 0 | 255 | 125.252 | 47.3165 |

## TWO DIMENSIONAL FEATURE SPACE

```
df_two_classes= pd.read_csv('...')

# Two features and plot the two-dimensional feature space
row1_index = 27260
row2_index = 27594

# Values of two features
row1_values = df_two_classes.iloc[row1_index, :].values
row2_values = df_two_classes.iloc[row2_index, :].values

x_coordinates = df_two_classes.columns

# Create a scatter plot for two features
plt.scatter(x_coordinates, row1_values, label=f'Feature
{row1_index}')
plt.scatter(x_coordinates, row2_values, label=f'Feature
{row2_index}', color='red')

plt.xlabel('Observations')
plt.ylabel('Feature Values')
plt.title('Two Feature Scatter Plot')
plt.legend()
plt.xticks(x_coordinates[::50])
plt.ylim(0, 220)
plt.show()
```

The code above, reads my CSV file that is the feature space of a bird. I have chosen two features to use for the scatter plot. I randomly selected the features for each bird.

## CARDINAL

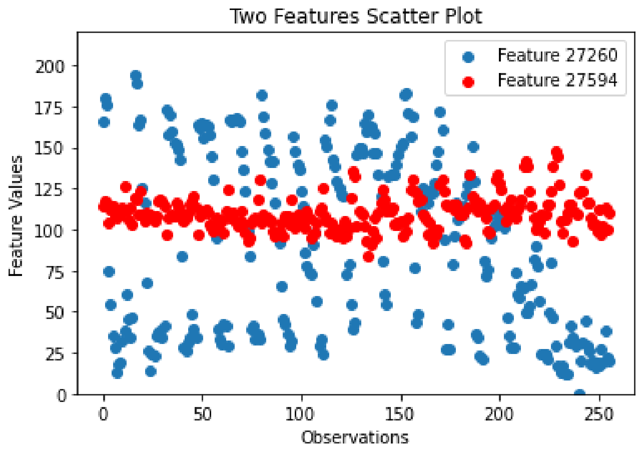


Fig. 8. Cardinal 2D Scatter plot

In the cardinal scatter plot, I noticed that the second feature is a strong feature as the value is in between 100-125. The first feature is almost the transpose of the first one as it the values tend to ignore that 100-125 threshold.

## SPARROW

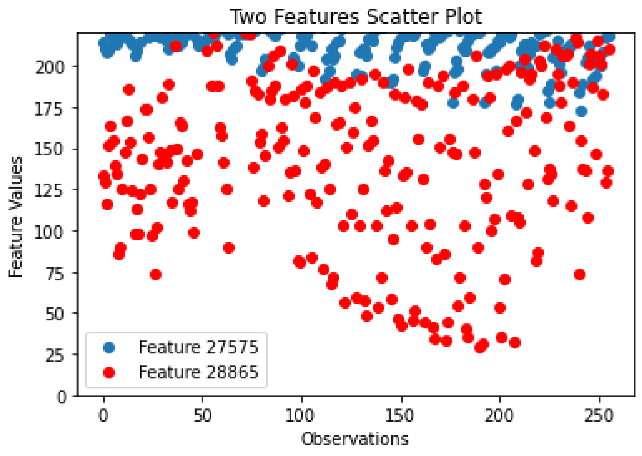


Fig. 9. Sparrow 2D Scatter plot

In the sparrow scatter plot, I noticed the first and second feature go higher than 200. It looks like the first feature tends to be the one that stays in the higher than 200. The second is more dispersed.

## CROW

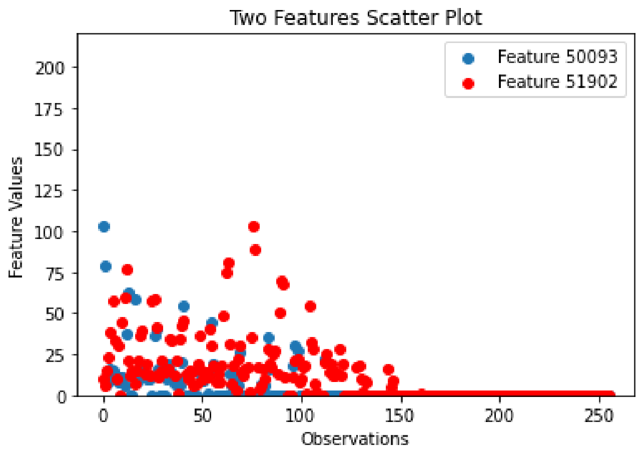


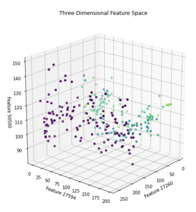
Fig. 10. Crow 2D Scatter plot

In the crow scatter plot, I believe that values are too low as almost none of the first feature shows up. Although they're all clustered on one side. Most values are 0-120.

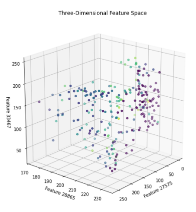
## THREE DIMENSIONAL FEATURE SPACE

I then read my CSV files for all birds and choose three features to use for the scatter plot. I randomly selected all three birds.

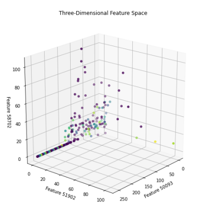
```
df_three_classes = pd.read_csv('...')

row1_index = 27260
row2_index = 27594
row3_index = 50500

# Get values of features
row1_values = df_three_classes.iloc[row1_index,:].values
row2_values = df_three_classes.iloc[row2_index,:].values
row3_values = df_three_classes.iloc[row3_index,:].values

x_coordinates = df_three_classes.columns

# Creates three-dimensional scatter plot
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')

sc = ax.scatter(x_coordinates,
row1_values,row2_values,
c=row3_values, cmap='viridis', marker='o')

ax.set_xlabel(f'Feature {row1_index}')
ax.set_ylabel(f'Feature {row2_index}')
ax.set_zlabel(f'Feature {row3_index}')
ax.set_title('Three-Dimensional Feature Space')

ax.view_init(elev=20, azim=40)
plt.xticks(x_coordinates[::50])
ax.set_box_aspect([1.5, 1.5, 1.5])

# Show the plot
plt.show()
```



(a) Cardinal 3D Scatter plot



(b) Sparrow 3D Scatter plot



(c) Crow 3D Scatter plot

Fig. 11. Three dimensional feature space for all three birds

## THREE DIMENSIONAL RESULTS

In the cardinal 3D scatter plot, I believe that values are too low as almost none of the first feature shows up. Although they're all clustered on one side. Most values are 0-120.

In the sparrow 3D scatter plot, all the points are scattered across the map. There are more high values at the end of the characteristics.

In the crow 3D scatter plot it seems that a lot of the values are low or 0. Although there are are dispersed across the map it's not clustered or gathered.

## RIDGE REGRESSION

In Ridge regression, we use a simple model y = Ax
A = yx'(xx' + $\lambda$ I)$^{-1}$

```
import pandas as pd
import numpy as np
from sklearn.metrics import confusion_matrix
# Ridge Classifier
sklearn.linear_model import Ridge Classifier

# Read a feature space
input_data1 = pd.read_csv("/Users/cristian/Desktop/
merged02.csv",header=None)

input_data = input_data1.copy()

NN = 255
```

The code shows that we first import pandas, numpy, confusion_matrix, Ridge Classifier. Then we read our merged feature space csv file. We copy the data that way we still have the original data. NN is a variable defined by the size of features in our csv file, in our case 256 but since it starts with 0 we put 255.

```
# Label/Response set
y = input_data[NN]

# Drop the labels and store the features
input_data.drop(NN,axis=1,inplace=True)
X = input_data

# Generates feature matrix using Numpy array
tmp = np.array(X)
X1 = tmp[:,0:NN]

# Generates label matrix using Numpy array
Y1 = np.array(y)

# Split the data into 80:20
row, col = X.shape

TR = round(row*0.8)
TT = row-TR
```

This is the train-test split. TR represents the training model. TT represents the testing model. It is split into 80:20, Training with 80% of the data and testing with 20%.

```
# Training with 80% data
X1_train = X1[0:TR-1,:]
Y1_train = Y1[0:TR-1]

ric = RidgeClassifier(alpha=0.01)
ric.fit(X1_train, Y1_train)

# Testing with 20% data
X1_test = X1[TR:row,:]
y_test = Y1[TR:row]

yhat_test = ric.predict(X1_test)
```

The code above is what trains the x and y with 80% of the data. Uses the ridge classifier with the trained X and trained Y. Then it tests with the remaining 20% of the data.

```
# Confusion matrix analytic
CC_test = confusion_matrix(y_test, yhat_test)

TN = CC_test[1,1]
FP = CC_test[1,0]
FN = CC_test[0,1]
TP = CC_test[0,0]

FPFN = FP+FN
TPTN = TP+TN
```

**REVISED**

TN represents the true negatives. TN is the amount where the model correctly predicted the negative class. FP represents the false positives. FP is the amount where the model incorrectly predicted the positive class. FN represents the false negative. FN is the amount where the model incorrectly predicted the negative class. TP represents the true positives. TP is the amount where the model correctly predicted the positive class. FPFN is (false positives + false negatives). TPTN is (true positives + true negatives). Both FPFN and TPTN will be used to predict the accuracy.

Then I would figure out the accuracy, precision, sensitivity, and specificity for all merged csv files. The following are the formulas for accuracy, precision, sensitivity, and specificity.

$$Precision = \frac{1}{(1 + (\frac{FP}{TP}))}$$
$$Sensitivity = \frac{1}{(1 + (\frac{FN}{TP}))}$$
$$Specificity = \frac{1}{(1 + (\frac{FP}{TN}))}$$
$$Accuracy = \frac{1}{(1 + (\frac{FPFN}{TPTN}))}$$

$$\frac{a}{\infty} = 0$$
$$\frac{a}{0} = \infty$$
$$\frac{0}{0} = undefined$$

If a precision = 1 , than the model is good because it has a higher TP than FP therefore it has greater precision.

The same applies to sensitivity. The amount of FP has to be lower than the TP if not it would influence the model. If FN were greater than TP then $(\frac{FN}{TP})$ is infinity and based on above, $\frac{a}{\infty} = 0$.

Specificity is no different you'd like for the false positives to be lower than the true negatives.

Typically if the data is imbalanced then accuracy is not going to be good. There could be a majority class and minority, it's always going to be biased towards the major. Example: if TP = $\infty$ and TN is very low, since $\infty$ is the majority class it's going to go off of that and could provide a false 1 for accuracy score.

Overall, what you'd like to see from your model is 1, 1, 1 for precision, sensitivity, and specificity.

## CARDINAL AND SPARROW RESULTS

Accuracy Score: 0.603165212246709
Precision Score: 0.6046864829681059
Sensitivity Score: 0.7639802631578948
Specificity Score: 0.41471249598458076

TN : 1291
FP : 1822
FN : 861
TP : 2787

## SPARROW AND CROW RESULTS

Accuracy Score: 0.6950970377936669
Precision Score: 0.6979785969084423
Sensitivity Score: 0.3844138834315652
Specificity Score: 0.8936793637505231

TN : 4270
FP : 508
FN : 1880
TP : 1174

## CARDINAL AND CROW RESULTS

Accuracy Score: 0.593494959193471
Precision Score: 0.5849420849420849
Sensitivity Score: 0.24897288414133112
Specificity Score: 0.8622089297158727

TN : 4036
FP : 645
FN : 2742
TP : 909

For all the merged csv files, I got more true negatives than false negatives. Same with true positives being greater than false positives.

## STANDARD REGRESSION

In standard regression, we use a simple model y = Ax. A=yx'(xx')$^{-1}$. The parameterized model helps achieve the classification objective.

```
import pandas as pd
import numpy as np
from numpy.linalg import inv
import matplotlib.pyplot as plt

# Reads feature space
input_data1 = pd.read_csv("/Users/cristian/Desktop/
merged.csv",header=None)

input_data = input_data1.copy()
```

We first import the necessary libraries, ie. pandas and numpy. Then read the feature space csv file. After reading the file we copy the data so that we can have still have the original.

```
y = input_data[64]

input_data.drop(64,axis=1,inplace=True)
X = input_data

X1 = np.array(X)

X2 = X1.transpose()
```

Y represents the Label/Response set. The labels are dropped and the features are stored. Then generates a feature matrix into a numpy array. X2 is the transpose of the numpy array.

```
XX = np.matmul(X2, X1)

IX = inv(XX)

TX = np.matmul(X1, IX)

Y1 = np.array(y)

Y2 = Y1.transpose()

A = np.matmul(Y2, TX)
```

In the code above, XX is the matrix product of the arrays X1 and X2. IX is the inverse of the square of the feature matrix. Then it matrix multiplies the X1 numpy array with the inverse square matrix and stores it into TX. Y1 then is created as a label matrix using numpy. We transpose Y1 and store it into Y2. A then is then the output of the matrix multiplication of Y2 and TX.

```
# Validating the model - compare it with y[1:5]
Z1 = np.matmul(X1[0,:], A)
Z2 = np.matmul(X1[1,:], A)
Z3 = np.matmul(X1[2,:], A)
Z4 = np.matmul(X1[3,:], A)
Z5 = np.matmul(X1[4,:], A)
```

**REVISED** We validate the model using the first 5 features. We do this by matrix multiplying X1 with A. Cardinal is labeled as 1, Sparrow is labeled as 2, and Crow is labeled as 3

## Cardinal & Sparrow

Z1 = 2.0734398429595005
Z2 = 0.5568419110295612
Z3 = 0.9553236530443177
Z4 = 1.3313904494548134
Z5 = 0.7926483011283343

Y1 = 2
Y2 = 1
Y3 = 1
Y4 = 1

## Cardinal & Crow

Z1 = 4.146679654529311
Z2 = 0.459182277914582
Z3 = 1.7781678007000834
Z4 = 0.2836463172646588
Z5 = 1.2315933504040162

Y1 = 3
Y2 = 3
Y3 = 1
Y4 = 3

## Crow & Sparrow

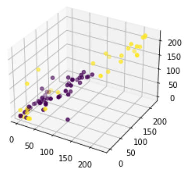
Z1 = 3.0574461288591355
Z2 = 0.5072765545582325
Z3 = 1.1935432771936088
Z4 = 0.5700686891159273
Z5 = 0.6625427065969118

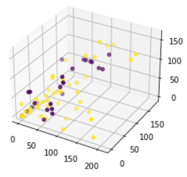
Y1 = 3
Y2 = 3
Y3 = 1
Y4 = 3

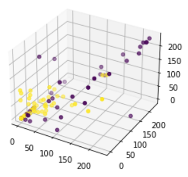
For almost all the Z's they reflect a good accuracy compared to their y labels. The only one I had a issue with is the crow and sparrow as their Z2, Z3, and Z4 don't seem to match the y label. This suggests that the transformation may not be effective in accurately predicting the labels from the given features.

**REVISED**

## ORIGINAL MERGED CSV


(a) Cardinal and Sparrow


(b) Cardinal and Crow


(c) Sparrow and Crow

## RANDOM FOREST

**REVISED**

```
# Same imports as before now with the addition of
# RandomForestClassifier
# code same as before ...
rF = RandomForestClassifier(random_state=0,
n_estimators=500, oob_score=True,n_jobs=-1)

model = rF.fit(X1_train,Y1_train)
importance = model.feature_importances_
indices = importance.argsort()[::-1]
```

We prepare the model using the Random Forest classifier.

```
oob_error = 1 - rF.oob_score_

# Testing with 20% data
X1_test = X1[TR:row,:]
y_test = Y1[TR:row]
yhat_test = rF.predict(X1_test)

# Confusion matrix analytics
CC_test = confusion_matrix(y_test, yhat_test)

# Rest of the code is the same.
```

The model is tested same as before with 20%. The yhat_test is now using the randomForest.predict().

### Cardinal and Sparrow Results

Our Accuracy Score: 0.9631711285312824
Our Precision Score: 0.9538050734312417
Our Sensitivity Score: 0.9791666666666667
Our Specificity Score: 0.9444265981368454
BuiltIn Accuracy: 0.9631711285312824
BuiltIn Precision: 0.9538050734312417
BuiltIn Sensitivity (recall): 0.9791666666666666

TN = 2940 FP = 173 FN = 76 TP = 3572

### Sparrow and Crow Results

Our Accuracy Score: 0.9705056179775281
Our Precision Score: 0.9611238157464881
Our Sensitivity Score: 0.9633267845448591
Our Specificity Score: 0.9750941816659691
BuiltIn Accuracy: 0.9705056179775281

TN = 4659 FP = 119 FN = 112 TP = 2942

Not sure what happens here for both precision and sensitivity but I get the following error: pos_label=1 is not a valid label. It should be one of [2, 3]

### Cardinal and Crow Results

Our Accuracy Score: 0.9597935669707154
Our Precision Score: 0.9646860986547084
Our Sensitivity Score: 0.9427554094768557
Our Specificity Score: 0.9730826746421706
BuiltIn Accuracy: 0.9597935669707153
BuiltIn Precision: 0.9646860986547086
BuiltIn Sensitivity (recall): 0.9427554094768557

TN = 4555 FP = 126 FN = 209 TP = 3442

We can see that the accuracy, precision, sensitivity, and specificity scores were much higher than when using ridge regression. So the model predicts the more accurately based on what the label should be. What I can infer would be that the RF model is better suited to capture the complexity of my data in relation to their features and target y label. We can see how for all merged csv files their TN and TP were higher significantly while keeping low of the FP and FN.

## PCA MODEL

### Cardinal & Sparrow

Our Accuracy Score: 0.9464576246117438
Our Precision Score: 0.9360403397027601
Our Sensitivity Score: 0.9668311403508772
Our Specificity Score: 0.9225827176357212

BuiltIn Accuracy: 0.9464576246117439
BuiltIn Precision: 0.9360403397027601
BuiltIn Sensitivity (recall): 0.9668311403508771

TN = 2872 FP = 241 FN = 121 TP = 3527

### 0.1 Sparrow & Crow

Our Accuracy Score: 0.9595250255362615
Our Precision Score: 0.9615514333895447
Our Sensitivity Score: 0.9335297969875572
Our Specificity Score: 0.9761406446211806

TN = 4664 FP = 114 FN = 203 TP = 2851

BuiltIn Accuracy: 0.9595250255362615

As I stated before I get an error when trying to use the built in precision and specificity on Sparrow and Crow.

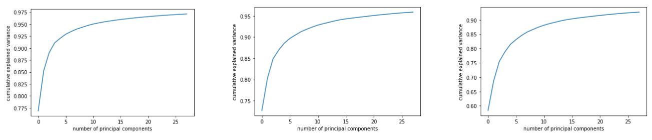
### Cardinal & Crow

Our Accuracy Score: 0.9477916466634662
Our Precision Score: 0.9663573085846868
Our Sensitivity Score: 0.9126266776225692
Our Specificity Score: 0.9752189703054904
TN = 4565 FP = 116 FN = 319 TP = 3332

BuiltIn Accuracy: 0.9477916466634662
BuiltIn Precision: 0.9663573085846868
BuiltIn Sensitivity (recall): 0.9126266776225691

For most of the scores went down when compared to when not applying Principal Component Analysis (PCA). I also noticed when I tested with higher dimensions that would affect the scores as well. Lowering them even more than what they are expected. I used 28 dimensions for all three merged csv files. The scores though were still higher than using ridge classification. We can also see that the TN and TP are significantly higher than FP and FN.

## PCA Plots



(a) (b) (c)

Fig. 13. a) cardinal and sparrow, b) sparrow and crow, c) cardinal and crow

The plots can help with determining the amount of principal components are needed to retain a good amount of variance. The plot basically demonstrates the relationship between the components and the level of variance that is accounted for.