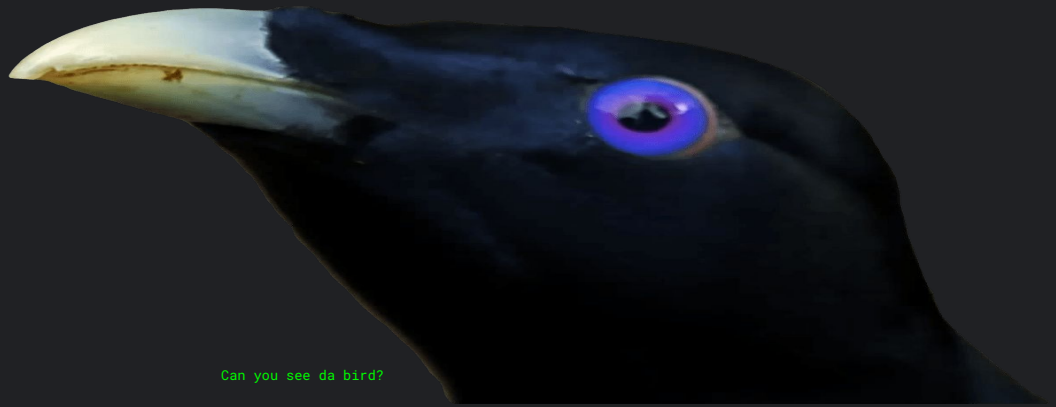
411 Group Assignment

Group 06a

Project 06 Bird species recognition



Can you see da bird?

Student Information:

Name:	ID:	Specific tasks done:	Slides number done:	Percentage of work done:
[redacted] Kit [redacted]	[redacted]	-coding Neural network -coding deep learning -research of models -formatting slides -comparison of models -visualisation of datasets	3-4,6-9,11-16,27-28,32-39,41,44	30
Cressensia [redacted]	[redacted]	-coding Neural network -coding deep learning -research of models -formatting slides -evaluation of results and datasets	5,17,29-32,40,42-47	30
Jeff [redacted] [redacted] [redacted]	[redacted]	-coding naive bayes	21-25	13
Heng Sheng [redacted]	[redacted]	-coding naive bayes	21-25	13
[redacted] Jing Zhi	[redacted]	-coding support vector machine	18-21	13
[redacted] Hui Ting	[redacted]	-ran the support vector machine code and read out slide 16		1

Content:

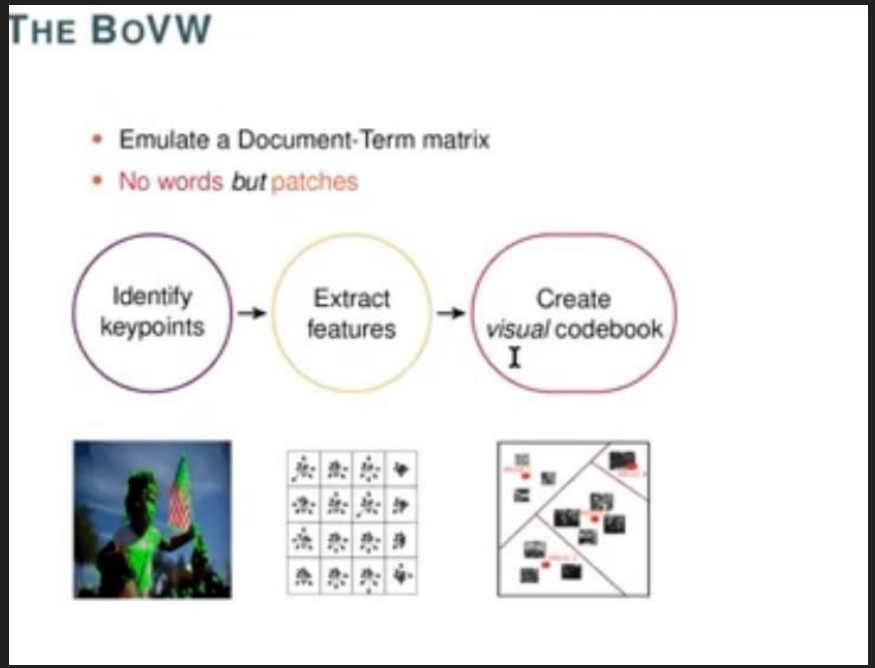
1. **Introduction** (define the problem and the goal)
2. **Methods** (propose approaches, and discuss their strengths and weaknesses & performance measures)
3. **Results** (Figures and tables of data analysis)
4. **Discussion** (discovered knowledge from data mining)

Introduction

Problem and goal:

Using extracted features from Caltech-UCSD Birds-200-2011 dataset to classify **200** different bird species.

BACKGROUND ON TRAINING.CSV & TESTING.CSV



Example of extracting features

We found out that our data have been pre-trained by the Convolutional Neural Network (CNN) ResNet-18 architecture.

The CNN model in simple basically “cuts” the images into grids and then compute the position of the target pixels into gradients.

These gradients correlated to the features of the input images that we can feed into our models.

Res-Net18 and feature extraction

- From the Res-Net18 docs:
  - To get the feature representations of the training and test images, use activations on the global pooling layer, 'pool5', at the end of the network. The global pooling layer pools the input features over all spatial locations, giving **512** features in total.

Layer Name	Output Size	ResNet-18
conv1	$112 \times 112 \times 64$	7×7, 64, stride 2
conv2_x	$56 \times 56 \times 64$	3×3 max pool, stride 2
		$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$
conv3_x	$28 \times 28 \times 128$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$
conv4_x	$14 \times 14 \times 256$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$
conv5_x	$7 \times 7 \times 512$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$
average pool	$1 \times 1 \times 512$	7×7 average pool
fully connected	1000	512×1000 fully connections
softmax	1000	

https://www.mathworks.com/help/deeplearning/ug/extract-image-features-using-pretrained-network.html

The dataset:

Structure:

Training set: **5993** observations with **514** features.

Testing set: **5793** observations with **514** features.

	X001.Black_footed_Albatross.Black_Footed_Albatross_0009_34.jpg	X1	X0.8839822	X0.07196508	X0.02295723	X0.005957174	X2.218124	X0.5668739
1	001.Black_footed_Albatross\Black_Footed_Albatross_0074_5...	1	0.494570600	0.80360410	0.306543300	0.7075512000	1.604019000	1.4195740
2	001.Black_footed_Albatross\Black_Footed_Albatross_0014_8...	1	1.649679000	1.19330700	0.164784400	0.5719254000	0.739176100	0.5143672
3	001.Black_footed_Albatross\Black_Footed_Albatross_0031_1...	1	1.465360000	2.01087200	1.761245000	0.6841311000	0.581619300	1.4693470
4	001.Black_footed_Albatross\Black_Footed_Albatross_0051_7...	1	0.485191000	0.57572590	0.174168300	0.2748663000	1.896859000	0.4925596
5	001.Black_footed_Albatross\Black_Footed_Albatross_0010_7...	1	0.631917100	0.08265538	0.032405340	0.7003007000	1.508773000	0.5100808
6	001.Black_footed_Albatross\Black_Footed_Albatross_0023_7...	1	1.848150000	1.20379300	1.730208000	0.3251984000	0.720037300	0.3659131
7	001.Black_footed_Albatross\Black_Footed_Albatross_0040_7...	1	0.175220200	0.74167490	0.029286340	0.1873556000	2.272518000	1.0792790
8	001.Black_footed_Albatross\Black_Footed_Albatross_0089_7...	1	0.001309960	0.09552376	0.082806840	0.2184793000	2.117950000	0.1866795
9	001.Black_footed_Albatross\Black_Footed_Albatross_0067_1...	1	0.737915200	1.91135000	1.091104000	0.7177411000	1.120939000	0.9042007
10	001.Black_footed_Albatross\Black_Footed_Albatross_0060_7...	1	0.168551100	0.53925560	0.001551453	0.2289452000	3.091724000	0.4949467
11	001.Black_footed_Albatross\Black_Footed_Albatross_0056_7...	1	0.969149500	0.05506871	0.360696900	0.5779266000	1.636767000	0.0021310
12	001.Black_footed_Albatross\Black_Footed_Albatross_0080_7...	1	0.534801300	0.24288200	0.023657210	1.2210170000	1.159858000	0.1812026
13	001.Black_footed_Albatross\Black_Footed_Albatross_0047_7...	1	2.932787000	1.24036300	0.024858610	0.3001463000	1.099381000	1.5241090
14	001.Black_footed_Albatross\Black_Footed_Albatross_0017_7...	1	0.430709000	0.54178830	0.572671800	0.4704633000	1.773097000	1.5266070
15	001.Black_footed_Albatross\Black_Footed_Albatross_0019_7...	1	2.592487000	2.44196300	0.114897300	0.7882829000	1.097469000	1.0056810
16	001.Black_footed_Albatross\Black_Footed_Albatross_0057_7...	1	0.620545000	1.08860400	0.000000000	0.6575109000	1.635850000	0.8356060
17	001.Black_footed_Albatross\Black_Footed_Albatross_0041_7...	1	0.003315040	0.87432260	0.000000000	1.4712430000	0.531633700	1.7486600
18	001.Black_footed_Albatross\Black_Footed_Albatross_0071_7...	1	0.585566700	1.23512600	0.137641400	0.0666374600	3.054998000	0.2943053

- **1st column** is str value name of the images.
- **2nd column** is the target classes in discrete data (1-200)
- **Column 3:514** are the features in continuous form

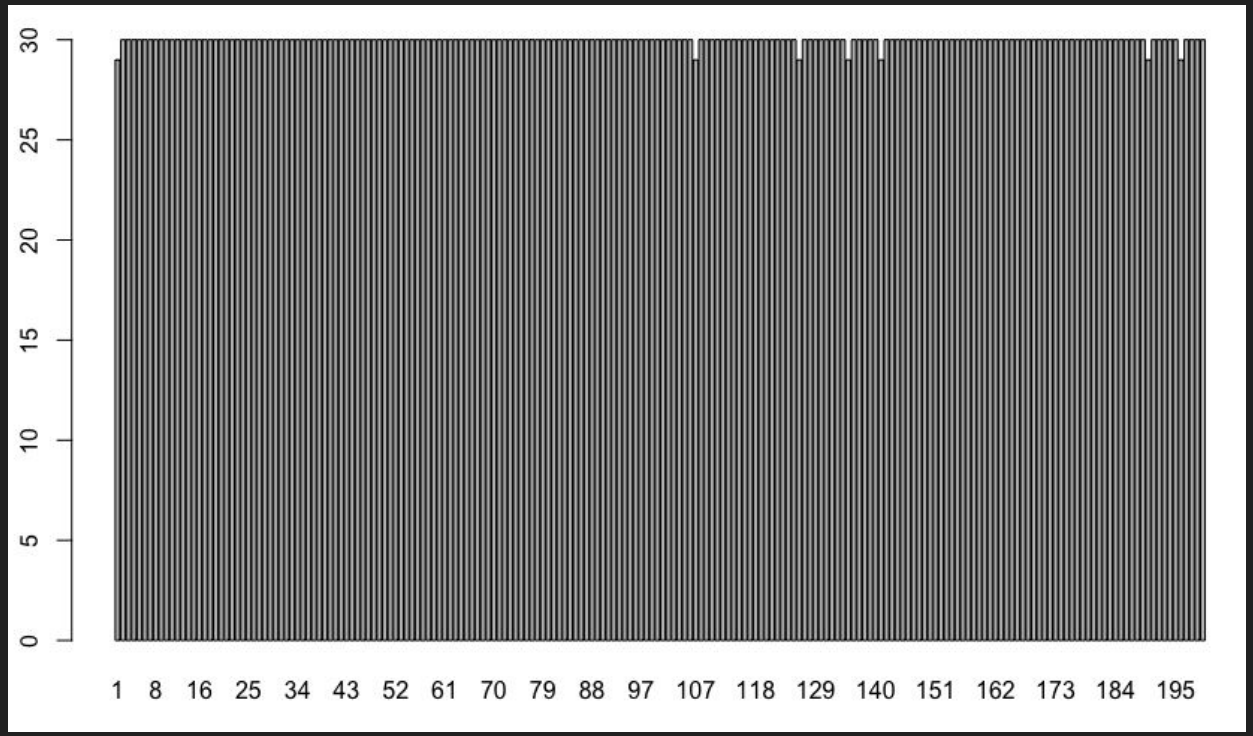
Potential Challenges:

X0.8839822	X0.07196508	X0.02295723	X0.005957174	X2.218124	X0.5668739

	X001.Black_footed_Albatross.Black_Footed_Albatross_0009_34.jpg	X1	X0.8839822	X0.07196508	X0.02295723	X0.005957174	X2.218124	X0.5668739
1	001.Black_footed_Albatross\Black_Footed_Albatross_0074_5...	1	0.494570600	0.80360410	0.306543300	0.7075512000	1.604019000	1.4195740
2	001.Black_footed_Albatross\Black_Footed_Albatross_0014_8...	1	1.649679000	1.19330700	0.164784400	0.5719254000	0.739176100	0.5143672
3	001.Black_footed_Albatross\Black_Footed_Albatross_0031_1...	1	1.465360000	2.01087200	1.761245000	0.6841311000	0.581619300	1.4693470
4	001.Black_footed_Albatross\Black_Footed_Albatross_0051_7...	1	0.485191000	0.57572590	0.174168300	0.2748663000	1.896859000	0.4925596
5	001.Black_footed_Albatross\Black_Footed_Albatross_0010_7...	1	0.631917100	0.08265538	0.032405340	0.7003007000	1.508773000	0.5100808
6	001.Black_footed_Albatross\Black_Footed_Albatross_0023_7...	1	1.848150000	1.20379300	1.730208000	0.3251984000	0.720037300	0.3659131
7	001.Black_footed_Albatross\Black_Footed_Albatross_0040_7...	1	0.175220200	0.74167490	0.029286340	0.1873556000	2.272518000	1.0792790
8	001.Black_footed_Albatross\Black_Footed_Albatross_0089_7...	1	0.001309960	0.09552376	0.082806840	0.2184793000	2.117950000	0.1866795
9	001.Black_footed_Albatross\Black_Footed_Albatross_0067_1...	1	0.737915200	1.91135000	1.091104000	0.7177411000	1.120939000	0.9042007
10	001.Black_footed_Albatross\Black_Footed_Albatross_0060_7...	1	0.168551100	0.53925560	0.001551453	0.2289452000	3.091724000	0.4949467
11	001.Black_footed_Albatross\Black_Footed_Albatross_0056_7...	1	0.969149500	0.05506871	0.360696900	0.5779266000	1.636767000	0.0021310
12	001.Black_footed_Albatross\Black_Footed_Albatross_0080_7...	1	0.534801300	0.24288200	0.023657210	1.2210170000	1.159858000	0.1812026
13	001.Black_footed_Albatross\Black_Footed_Albatross_0047_7...	1	2.932787000	1.24036300	0.024858610	0.3001463000	1.099381000	1.5241090
14	001.Black_footed_Albatross\Black_Footed_Albatross_0017_7...	1	0.430709000	0.54178830	0.572671800	0.4704633000	1.773097000	1.5266070
15	001.Black_footed_Albatross\Black_Footed_Albatross_0019_7...	1	2.592487000	2.44196300	0.114897300	0.7882829000	1.097469000	1.0056810
16	001.Black_footed_Albatross\Black_Footed_Albatross_0057_7...	1	0.620545000	1.08860400	0.000000000	0.6575109000	1.635850000	0.8356060
17	001.Black_footed_Albatross\Black_Footed_Albatross_0041_7...	1	0.003315040	0.87432260	0.000000000	1.4712430000	0.531633700	1.7486600
18	001.Black_footed_Albatross\Black_Footed_Albatross_0071_7...	1	0.585566700	1.23512600	0.137641400	0.0666374600	3.054998000	0.2943053

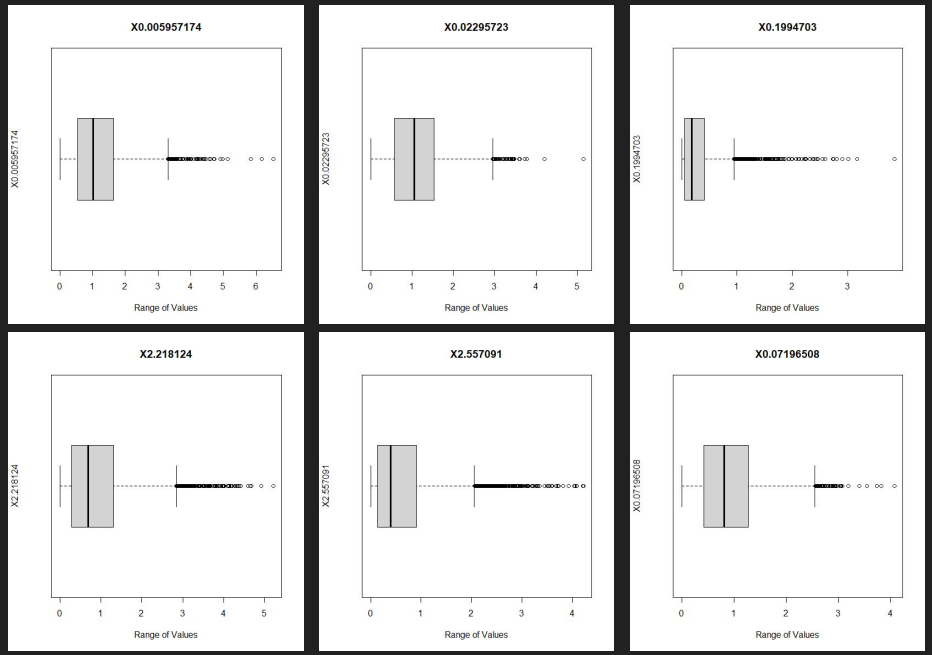
- **200** Classes
- The dataset provided only has **~30** observations for each class.
- The features are ambiguous ie: Features have no intrinsic meaning
- Large data set
- Some values are very close to 0

Potential Challenges:



- **200** Classes
- The dataset provided only has **~30** observations for each class.
- The features are ambiguous ie: Features have no intrinsic meaning
- Large data set
- Some values are very close to 0

Dataset:



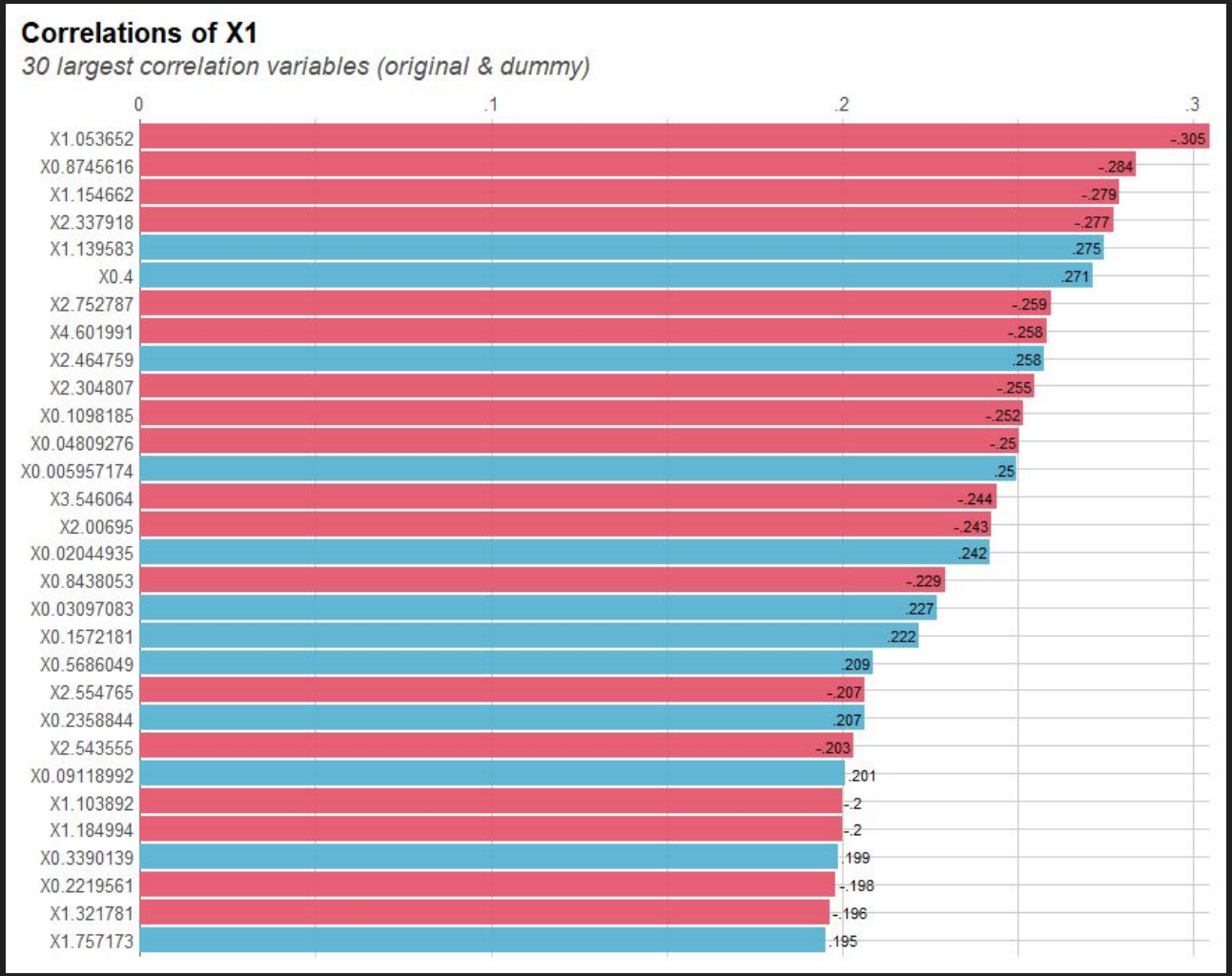
- To get a feel for the features:
- Box plots for first 6 features

- Most of the data are clustered around 0-1 with significant amount of outliers

Relationships between data

Unranked Correlation coefficients:

	[,1]
X1	1.00000000000
X0.8839822	0.10548254229
X0.07196508	0.03468865350
X0.02295723	0.03952805719
X0.005957174	0.24987193320
X2.218124	-0.15496667053
X0.5668739	0.02269815674
X0.8199308	0.10703771695
X2.721439	-0.17127366730
X1.557017	0.04189392359
X1.936072	-0.18021134461
X1.68093	0.10625607822
X1.710006	-0.01540456095
X5.365936	-0.17890006616
X0.5047694	0.08131737411


Correlations of X1
30 largest correlation variables (original & dummy)
0
.1
.2
.3
X1.053652
-.305
X0.8745616
-.284
X1.154662
-.279
X2.337918
-.277
X1.139583
.275
X0.4
.271
X2.752787
-.259
X4.601991
-.258
X2.464759
.258
X2.304807
-.255
X0.1098185
-.252
X0.04809276
-.25
X0.005957174
.25
X3.546064
-.244
X2.00695
-.243
X0.02044935
.242
X0.8438053
-.229
X0.03097083
.227
X0.1572181
.222
X0.5686049
.209
X2.554765
-.207
X0.2358844
.207
X2.543555
-.203
X0.09118992
.201
X1.103892
-.2
X1.184994
-.2
X0.3390139
.199
X0.2219561
-.198
X1.321781
-.196
X1.757173
.195

Our approach:

Classification models:

1. Support Vector Machine
2. Naive Bayes classifier
3. Feedforward Neural Network and Deep Learning

Metric:

- Accuracy, mse, hit ratios

Reason:

- Most packages don’t offer ROC in large scale multinomial classifications

- Accuracy is the number of all correct predictions divided by the total number

- MSE measures the average of the squares of the errors

- Hit ratios (h2o) refers to the number of times that a correct prediction was made in ratio to the number of total prediction names.

Our approach:

-Use models that handle high-dimensional feature spaces

-Feature preprocessing & strategy:

- Scale features (0-1)
- Makes the data more generalizes and “closer”

- Using all features in the data set to train

	X0.8839822	X0.07196508	X0.02295723	X0.005957174	X2.218124	X0.5668739	X0.8199308	X2.721439	X1.557017	X1.936072	X1.68
1	0.1029142958	0.1971692035	0.0596000922	1.084643e-01	0.3078914484	0.2967773686	0.1474038910	0.345853378	0.272284115	0.0155707386	0.343
2	0.3432786998	0.2927852044	0.0320384280	8.767353e-02	0.1418848530	0.1075340518	0.0723174596	0.208985431	0.244807861	0.1040730360	0.215
3	0.3049240946	0.4933798005	0.3424324213	1.048741e-01	0.1116418251	0.3071829550	0.2801176089	0.329280435	0.290532838	0.1777551454	0.061
4	0.1009625119	0.1412578870	0.0338629053	4.213574e-02	0.3641020867	0.1029749361	0.0509590845	0.345028746	0.124878003	0.2737035879	0.175
5	0.1314944789	0.0202800053	0.0063004517	1.073529e-01	0.2896089787	0.1066379333	0.0982370372	0.479899079	0.038405813	0.2398456563	0.410
6	0.3845781688	0.2953580090	0.3363980109	4.985142e-02	0.1382111604	0.0764981093	0.3459866538	0.557793499	0.476935382	0.0528783213	0.073
7	0.0364612524	0.1819744938	0.0056940359	2.872075e-02	0.4362098321	0.2256350015	0.2584465185	0.500571851	0.272307276	0.4441742199	0.327
8	0.0002725872	0.0234373414	0.0160998309	3.349187e-02	0.4065405043	0.0390273778	0.0151449115	0.382140580	0.020019104	0.2540762315	0.131
9	0.1535514306	0.4689614663	0.2121393586	1.100264e-01	0.2151642420	0.1890329806	0.2939687054	0.281085128	0.224029972	0.8677966643	0.053
10	0.0350734915	0.1323096748	0.0003016433	3.509625e-02	0.5934564245	0.1034739853	0.0639007080	0.434979791	0.110984934	0.1847086026	0.085
11	0.2016685551	0.0135114464	0.0701289785	8.859349e-02	0.3141774271	0.0004455217	0.1131474007	0.498984638	0.180168539	0.4856491362	0.098
12	0.1112858289	0.0595925910	0.0045995848	1.871763e-01	0.2226347441	0.0378823723	0.1152686600	0.390198899	0.072223032	0.4974903735	0.277
13	0.6102783075	0.3043306832	0.0048331686	4.601105e-02	0.2110261839	0.3186315461	0.1338460298	0.233939253	0.299206385	0.0621343580	0.187
14	0.0896254517	0.1329310883	0.1113424828	7.211986e-02	0.3403459707	0.3191537802	0.2248550520	0.430288135	0.495968994	0.2463572183	0.234
15	0.5394659001	0.5991506261	0.0223390617	1.208401e-01	0.2106591755	0.2102485399	0.1236123858	0.296964311	0.202330827	0.2872571688	0.440
16	0.1291280793	0.2670956801	0.0000000000	1.007934e-01	0.3140014089	0.1746925133	0.1168102616	0.297787095	0.126373737	0.1943085231	0.263
17	0.0006898206	0.2145204220	0.0000000000	2.255348e-01	0.1020470892	0.3655763724	0.1619854755	0.306573039	0.088843514	0.5490444246	0.464
18	0.1218495085	0.3030457530	0.0267611138	1.021522e-02	0.5864068687	0.0615277206	0.1272967418	0.476638043	0.415404548	0.1507331670	0.229

Approach and methodology

- We knew that the SVM and Neural Networks were in our favour since the data have already been trained in the CNN with ResNet-18 architecture

- Therefore, for the last model, we wanted to test the limits of the CNN ResNet-18 with Naive Bayes classification.
- We decided to work with the neuralnet package as it offers

Support Vector Machine

- SVM is a popular machine learning algorithm for classification problems
- Able to extract features from CNN and use them to train a SVM classifier

Strengths:

- SVMs are particularly useful when the **number of features is high** and you have a relatively **small number of samples.**

Weaknesses:

- SVM is not suitable for large data sets.
- SVM does not perform very well when the data set has more noise i.e. target classes are overlapping

Support Vector Machine

```
svm_model = svm(
      formula = label~.,
      Data = train,
      Type = “C-Classification”
      Kernel = “radial”
)

test_pred = predict(svmModel, newdata = test)
test_pred

# Create a confusion matrix to evaluate model
performance
cm <- table(test$label, test_pred)

# Calculate the accuracy of the model
acc <- sum(diag(cm)) / sum(cm)
acc
# Print the accuracy
cat("Accuracy:", round(acc, 4), "\n")
```

```
>> Accuracy: 0.4739
```

Support Vector Machine (Fine tuning)

Challenges: SVM fine tuning took way too long, as a result, we had to reduce the amount of features used to optimize the time spent. We just need to ensure that the model is able to generalize and predict for unseen data (test set).

```
#set top 100 high correlation features as dataset features
top_cor_cols <- row.names(head(corTable, 100))
Train <- train[, c("label", top_cor_cols)]
Test <- test[, c("label", top_cor_cols)]

# tune SVM hyperparameters using cross-validation
svm_model <- train(label ~ ., data = train, method = "svmRadial",
                   trControl = trainControl(method = "cv", number =
3),
                   preProcess = c("center", "scale"))
# evaluate performance on test data
test_pred <- predict(svm_model, newdata = test)

# Create the confusion matrix
confusion <- table(test$label, test_pred)
confusion

# Compute the accuracy
accuracy <- sum(diag(confusion)) / sum(confusion)
accuracy
```

```
                [,1]
label 1.0000000000
x476  0.3047149218
x417  0.2837552781
x117  0.2791952000
x94   0.2778718589
x180  0.2749415700
x240  0.2716447262
x295  0.2598029671
x130  0.2591082282
x19   0.2579324697
x500  0.2553531505
x145  0.2513427663
x246  0.2501840755
x5    0.2501808320
x58   0.2446960265
x416  0.2431422365
x490  0.2424425983
x447  0.2293931269
```

```
>> Accuracy: 0.4748015
```

Support Vector Machine

There are several reasons why support vector machine took such a long time to run:

-Computational Complexity : SVM is known to have high computational complexity especially for larger dataset. When number of data and features increase, the time and resources required to train SVM also increase.

-**Memory usage** : SVM consume a lot of memory to store the data and the kernel matrix for large dataset. When our computer memory is limited, it will slow down the training process or even crash.

Support Vector Machine (Fine tuning)

Hyperparameters explanation:

Cost (C)

- This parameter control the penalty for misclassifying training example.
- A large value of Cost parameter (C) indicates poor accuracy but low bias

Gamma

- This parameter controls the shape of decision boundary that SVM creates to separate different classes in data
- High value of Gamma leads to more accuracy but biased results

Naive Bayes Classifier

- Naive Bayes classifier is a probabilistic machine learning model that’s used for classification task.
- The crux of the classifier is based on the Bayes theorem.

Strengths:

- Naive Bayes is fast and easy to implement allowing it to become a popular first choice among others.

Weaknesses:

- The biggest weakness in Naive Bayes is that the requirement of predictors to be independent. For most of the real life cases, the predictors are dependent, which will slightly hinder its performance

Naive Bayes Classifier

```
nb <- naiveBayes(traind[, -200], traind$predicton) #train the model
pred <- predict(nb, newdata = testd)#predict with test data
cm <- table(testd$predicton, pred)

# Calculate accuracy
accuracy <- sum(diag(cm)) / sum(cm)
cat("Accuracy:", round(accuracy, 3))
```

```
>> Accuracy:  0.612
```

Naive Bayes Classifier (Fine Tuning)

```
# Define cross-validation method
ctrl <- trainControl(method = "cv", number = 10, classProbs = TRUE)

# Define hyperparameter grid
tuneGrid <- expand.grid(fL = c(0, 0.5, 1), usekernel= c(TRUE) ,adjust = c(0, 0.5, 1))
```

← we first define method for cross-validation and the number of folds we will be executing

← Next we list out the hyperparameters the function will attempt to use and choose

Naive Bayes Classifier (Fine Tuning)

Hyperparameters explanation:

```
fL = c(0, 0.5, 1)
```

- Refers to Laplace Correction (fL, numeric)
- is a smoothing technique that handles the problem of zero probability in Naïve Bayes.
- Using higher alpha values will push the likelihood towards a value of 0.5

```
usekernel= c(TRUE)
```

- This parameter allows us to use a kernel density estimate for continuous variables versus a gaussian density estimate,

```
adjust = c(0, 0.5, 1)
```

- This parameter allows us to adjust the bandwidth of the kernel density, bigger numbers allow for a more flexible density estimate

Naive Bayes Classifier (Fine Tuning)

```
# Perform hyperparameter tuning
set.seed(123)
nb_tuned <- train(x = traind[, -ncol(traind)], y = traind[, ncol(traind)],
                  method = "nb", trControl = ctrl, tuneGrid = tuneGrid)
```

← We will then run the hyperparameter tuning with the values defined earlier, it will attempt to search for the best parameters from the range given earlier

```
# Train final model with optimal hyper parameters
nb_model <- naiveBayes(x = traind[, -ncol(traind)], y = traind[, ncol(traind)],
                       laplace = nb_tuned$bestTune$fL, adjust = nb_tuned$bestTune$adjust)
```

← After running the hyperparameter tuning we will attempt to train a new model with the best values acquired by the algorithm earlier

Naive Bayes Classifier (Fine Tuning)

```
# Get predicted class labels for test set
pred <- predict(nb_model, newdata = testd)
cm <- table(testd$predicton, pred)
# Calculate accuracy
accuracy <- sum(diag(cm)) / sum(cm)
cat("Accuracy:", round(accuracy, 3))
```

```
>> Accuracy:  0.625
```

Neural network (deep learning)

- A network consisting of thousands of connected nodes that takes in inputs and applies a weight & activation function to produce an output
- Can be used to classify data after it has been trained via a CNN

Strengths:

- Good performance / accuracy
- Handles large datasets well

Weaknesses:

- Takes long time to tune
- artificial neural networks are pretty much concealed in their actual structure
- Which may make it challenging to tune correctly

Feed Forward Neural Network

```
model = neuralnet(
        train_y~.,
        train,
        hidden=c(264, 200),
        threshold=0.04,
        learningrate = 0.1,
        algorithm = "backprop",
        lifesign = "full",
        lifesign.step = 1000,
        rep = 2,
        linear.output = FALSE
)
```

```
>> Accuracy: 0.51
```

Feed Forward Neural Network

Hyperparameters explanation:

```
hidden=c(264, 200)
```

- Refers to the number of hidden layers & nodes in each layer
- 2 hidden layers with 264 and 200 nodes.

```
threshold=0.04
```

- numeric value specifying the threshold for the partial derivatives of the error function as stopping criteria

```
learningrate = 0.1
```

- The learning rate controls how quickly the model is adapted to the problem.

```
algorithm = "backprop"
```

- Specifies training function
- taking the error rate of a forward propagation and feeding this loss backward through the neural network layers to fine-tune the weights

Feed Forward Neural Network (Fine tuning)

```
# Create a parameter grid for the hyperparameters to
be tuned

fine_tuned_model <- neuralnet
(
  train_y ~.,
  train_x_norm,
  hidden=c(100, 50),
  threshold=0.04,
  learningrate = 0.01,
  algorithm = "backprop",
  lifesign = "full",
  lifesign.step = 1000,
  rep = 2,
  linear.output = FALSE,
  startweights = model$weights
)
```

Feed Forward Neural Network (Fine tuning)

```
hidden: 264, 200    thresh: 0.04    rep: 1/2    steps:     11  error: 29694.67988     time: 34.52 secs
Error: vector memory exhausted (limit reached?)
```

Packages such as keras, neuralnet, and caret requires us to run the codes based on our CPU and memory. Our laptops cannot tank this.

We tried making our models less complex, and reduced the number of nodes but we don’t have enough computing power still.

SOLUTION - h2o package

We decided to further validate our findings using deep learning offered in the h2o package. As it can support large multinomial classification models WITHOUT relying on our own computing power. We can connect to their servers and use theirs instead.

We trained our models and then did fine tuning using deep learning in h2o.

Feed Forward Neural Network fine tuning

(with h2o package)

h2o package:

- H2O is the scalable open source machine learning platform that offers parallelized implementations of many supervised and unsupervised machine learning algorithms, such as: **Deep Learning**
- When the user makes a request, R queries the server via the REST API, which returns a JSON file with the relevant information that R then displays in the console

h2o advantages:

- Tuning time is faster as no actual data is stored in R.
- Takes about 10 minutes to do cartesian grid search
- We can use their processing power to run the codes. Hence we can work with large multinomial classification tasks.
- Many hyperparameters that we can tune
- It can support model regularisation to minimize the adjusted loss function and prevent overfitting or underfitting.
- Can perform cross validation on training set

Feed Forward Neural Network fine tuning

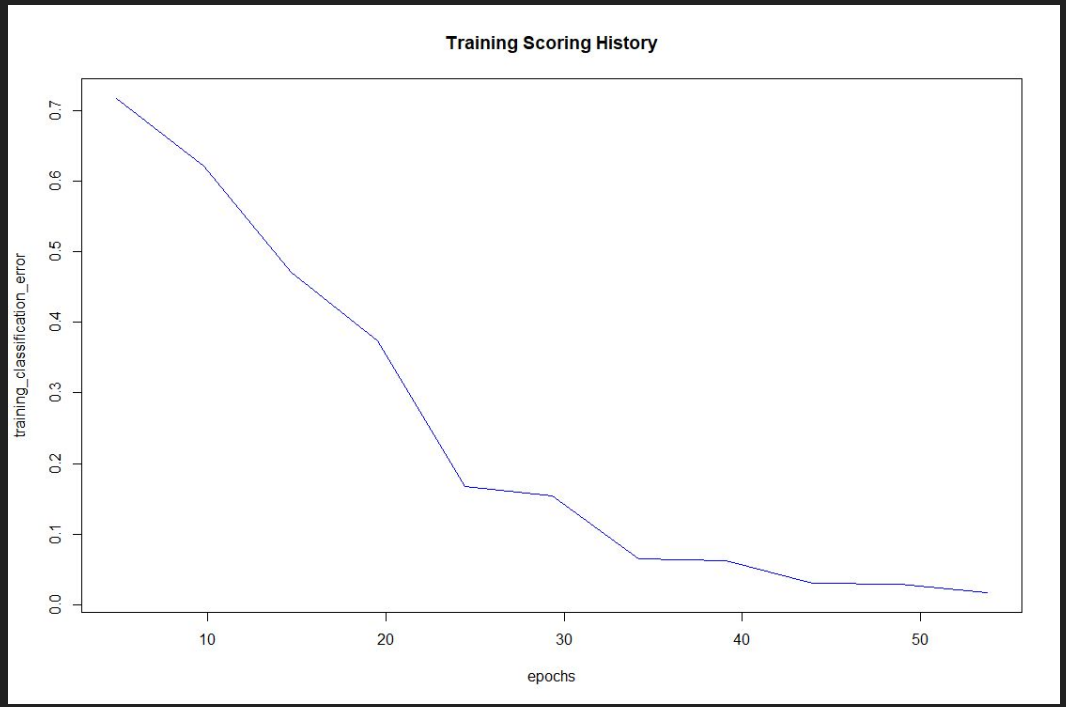
(with h2o package)

Create a simple deep learning model and train it: (default values)

```
dl_fit3 <- h2o.deeplearning(x = x,
                            y = y,
                            training_frame = as.h2o(dataset_training),
                            model_id = "dl_fit3",
                            epochs = 50,
                            hidden = c(200,200),
                            nfolds = 3,
                            #used for early stopping
                            score_interval = 1,
                            #used for early stopping
                            stopping_rounds = 5,
                            #used for early stopping
                            stopping_metric = "misclassification",
                            #used for early stopping
                            stopping_tolerance = 1e-3,
                            #used for early stopping
                            variable_importances = T,
                            seed = 1)
```

Feed Forward Neural Network fine tuning

(with h2o package)



Feed Forward Neural Network fine tuning

(with h2o package)

MSE from prediction with new test data:

```
> h2o.mse(dl_fit3)
[1] 0.01525391
```

Mean per class error with train data:

```
> h2o.mean_per_class_error(dl_fit3)
[1] 0.01800575
```

Mean per class error with test data:

```
> h2o.mean_per_class_error(dl_perf3)
[1] 0.9969815
```

Comments and analysis:

- Although the model performed well with the train set, we realised it is overfitted and did not do well with new data.
- The goal of this fine tuning is to **generalize the model** to new, unseen data. We want to achieve better performance on the new task while still maintaining the ability to generalize to unseen data

Feed Forward Neural Network fine tuning

(with h2o package)

Setting up the hyperparameter grid:

```
dl_params1 <- list( l2 = c(0, 0.0001, 0.001, 0.1, 1),
                    activation = c("Rectifier", "Tanh", "Rectifier with
dropout",
                                   "Tanh with dropout"),
                    epochs = c(10,20,30)
                    )
```

Fine tuning:

```
dl_grid <- h2o.grid("deeplearning", x = x, y = y,
                      grid_id = "dl_grid_tuned",
                      training_frame = as.h2o(dataset_training),
                      seed = 1,
                      balance_classes = TRUE,
                      train_samples_per_iteration = 0,
                      hyper_params = dl_params1)
```

```
>> mean per class error:
```

Feed Forward Neural Network fine tuning

(with h2o package)

Hyperparameters:

```
l2 = c(0, 0.0001, 0.001, 0.1, 1)
```

- In an effort to generalise the model, we tried different L2 regularization weights.
- L2 regularization uses the sum of the squared values of the weights.

```
activation = c("Rectifier",
               "Tanh",
               "Rectifier with dropout",
               "Tanh with dropout")
```

- Tanh activation function: (s-shaped) takes any real value as input and outputs values in the range -1 to 1
- Rectifier activation function: it changes any negative values to zero and has a straight line shape over the defined functional space.

Feed Forward Neural Network fine tuning

(with h2o package)

Best parameters
from fine tuning:

```
H2O Grid Details
================

Grid ID: dl_grid_8
Used hyper parameters:
  -  activation
  -  epochs
  -  l2
Number of models: 60
Number of failed models: 0

Hyper-Parameter Search Summary: ordered by decreasing mean_per_class_error
            activation   epochs      l2          model_ids mean_per_class_error
1            Rectifier 10.00000 1.00000 dl_grid_8_model_49              0.99500
2                 Tanh 10.00000 1.00000 dl_grid_8_model_50              0.99500
3 RectifierWithDropout 10.00000 1.00000 dl_grid_8_model_51              0.99500
4      TanhWithDropout 10.00000 1.00000 dl_grid_8_model_52              0.99500
5            Rectifier 20.00000 1.00000 dl_grid_8_model_53              0.99500

---
   activation   epochs      l2          model_ids mean_per_class_error
55  Rectifier 10.00000 0.00010 dl_grid_8_model_13              0.08783
56  Rectifier 30.00000 0.00010 dl_grid_8_model_21              0.07583
57  Rectifier 20.00000 0.00010 dl_grid_8_model_17              0.06300
58  Rectifier 10.00000 0.00000  dl_grid_8_model_1              0.06218
59  Rectifier 20.00000 0.00000  dl_grid_8_model_5              0.00433
60  Rectifier 30.00000 0.00000  dl_grid_8_model_9              0.00017
```

Feed Forward Neural Network fine tuning

(with h2o package)

```
# don't take the "best" model because it can cause
overfitting
# instead, take average performing model with
# mean per class error < 0.2.

We leveraged on the validation set which took an
average of mean per class error of <0.2.
```

Make predictions with best model from grid search:

```
best_dl_model <- h2o.getModel(dl_gridperf_mean@model_ids[[50]])
```

```
dl_perf <- h2o.performance(best_dl_model, newdata = as.h2o(dataset_testing))
```

Feed Forward Neural Network fine tuning

(with h2o package)

Confusion matrix
(best model and test set):

Comments and analysis:

From the analysis here, we can see that most of the predicted values were wrong. Also the hit ratios are consistently low. Hence we can conclude that our model is not working well. Low hit ratios could suggest the model is consistently missing certain patterns or features in the data. This can be an indication that the model needs to be improved, or that there may be certain factors that are not being accounted for in the data. The model might struggle to predict some values due to this. Overall we are quite happy that we are able to achieve this. We will explain why we are happy with this in the evaluations of the datasets.

Top-10 Hit Ratios:

	k	hit_ratio
1	1	0.004143
2	2	0.011393
3	3	0.016399
4	4	0.022441
5	5	0.026411
6	6	0.032108
7	7	0.036251
8	8	0.039013
9	9	0.041775
10	10	0.047298

Comparison of Models

model/ criteria	Performance	Time complexity & optimization	Performance with fine tuning
Naive Bayes (NB)	Accuracy: 0.612	Naive Bayes predicts fast and is simple to understand but tuning takes a long time	Accuracy: 0.625
Support vector machine (SVM)	Accuracy: 0.473	Efficiency of models like svm are more suitable in small dataset	Accuracy: 0.44
Deep Learning	Accuracy: 0.51	Deep learning handles large data set better but tuning is challenging	Accuracy: 0.1

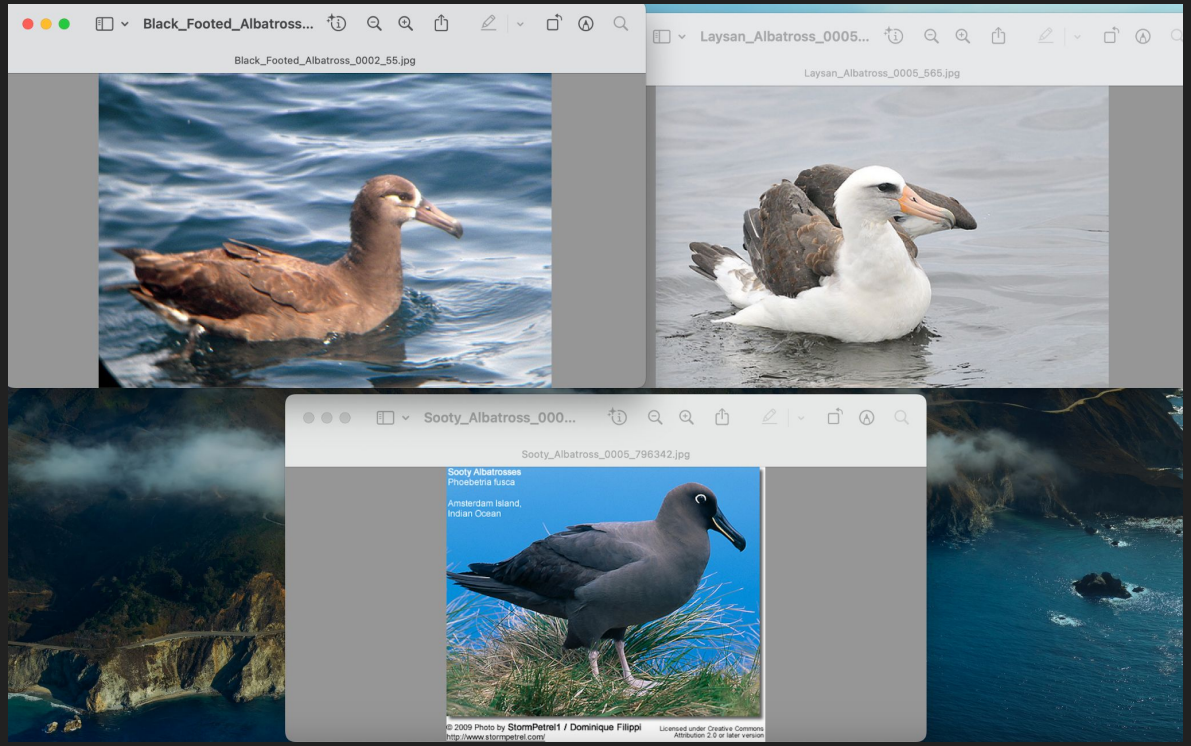
Although, we do have to mention that fine tuning across all models took a very very long time, and for a small amount of increase, it is not efficient.

Evaluation of results and datasets

In theory, we should be able to classify the birds quite accurately using deep learning models… BUT why doesn't it work?!

We only know that the architecture they used to pre-train the dataset is resnet-18. However, we suspect that there might be parameters like colours being overlooked by the models. For instance in this example, these birds belong to different groups. They have about the same beak , body, and neck shape, as each other and the absolute difference is in their colours. Our models might not be able to classify more precisely because of this issue. It can at most classify the most likely of the 3 birds shown. This issue is prevalent amongst other bird species too.

ofc this is just a hypothesis"

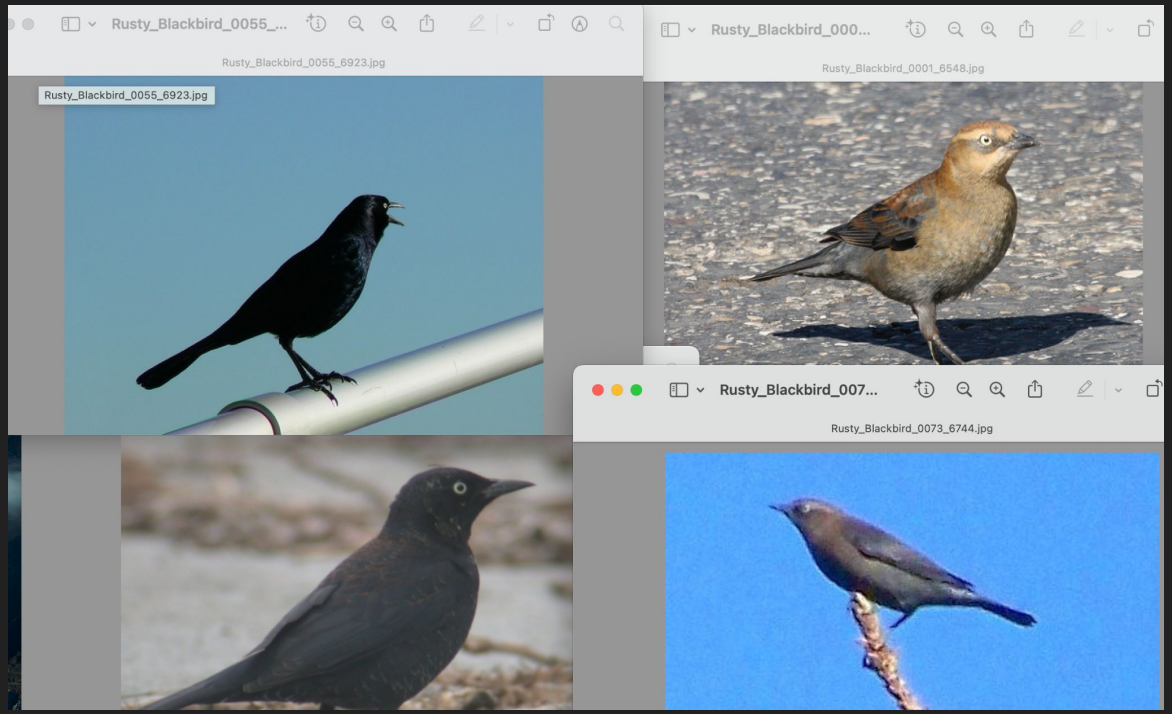


Evaluation of results and datasets

Why did naive bayes perform better than expected?

It could be that neural networks are by comparison more complex. There is many topology, decision in the hidden layers, and variants. Also neural networks need to be retrained after a single instance. However it does not guaranteed optimal. This makes it challenging to train with neural networks.

SVM also take into accounts of interaction between features to a certain degree.While features are treated as independent in naive bayes.



Discussion- Knowledge gained

1. Models all tend to perform poorly with new data.

- This can be due to vastly different data in train/test sets.
- Example: birds that are categorized in the same species can have different color ways for male and females
- For some bird species, the absolute difference is in their colours,eye shape,beak shape and tail size. Also in their activity patterns(diet, habitat, nocturnal/diurnal)which our datasets did not account for.

Example: (taken from dataset)
Bird of the same species but with different colors: shiny cowbird

Discussion- Knowledge gained

2. Understanding nuances of pre-trained data pictures

- After diagnosing some pictures of the birds, it is evident that there is inconsistency in the background surrounding the birds. We might not know if it affected the quality of the models as some background images could be translated into the gradients we use in our datasets
- Also found some pics with multiple birds, only certain parts of the birds, etc
- Inconsistency in number of datasets per bird could also be a factor of incorrect classification

taken from dataset

Discussion- Knowledge gained

3. Machine learning is an incremental process

- Many times we might blame the data for inaccuracy but it could also mean that we have not yet reached the capabilities of being able to train our models to perfection
- There are many things to explore and learn, new techniques to discover along the way to be able to tune our models better
- Some models are more sensitive to parameters as such if we are able to use the correct values, we might get much more accurate results.
- Good training accuracy does not mean good testing accuracy
- ALSO learned the hard way that computing power is so important definitely understand why high CPU and memory computers exist now
- There are many models and packages available for the same problem, so don’t limit to one.

FIN

Thank you!