Models:

***Explored*:** Naive Bayes, Gradient Boosted Trees, Decision Trees and Random Forest

***Observations*:**

Naive Bayes:

This was the first algorithm we tried to run for simple probabilistic classification. With prior probabilities, we tried to estimate a likelihood ratio and posterior probability. Using these we tried to predict and the results were poor. This is because Naive Bayes does not perform well when the data consists of extensive and negative continuous features. Due to this limitation, we selected a few minimal features resulting in an accuracy of approx. 46%

Gradient Boosted Trees:

We observed the pattern of this algorithm and although, it does try to account for loss of one tree into the next one as well as the validation tolerance for termination, we did not feel this would perform the best from a distributed computing perspective. Also, few observations led us to believe that as we increased the depth of the tree, our model started to overfit and that's due to the nature of its loss calculating functions. We achieved an accuracy of approx. 75% on validation.

PS: We did not look into deeper concepts of Multinomial and Bernoulli Naive Bayes classifiers.

Decision Trees:

We tried the single decision tree model with all the features and observed good results. The only drawback was that it didn't have the distributed property as all the features were used in a single tree. The accuracy was approx. 78%.

***Random Forest:* (Selected)**

We selected random forest simply because of its computational performance in a distributed environment. The individual trees could train in parallel by averaging across the probabilities in the end and thereby, leading to a better prediction than any of our previous models.

The following parameters were considered as well as tuned for increasing the accuracy of our predictions:

Number of Trees: Minimum number of decision trees to be constructed.
Feature subset strategy: Minimum number of features to be used per tree selected at random.
Impurity: Measure of the homogeneity of the target variable within the subsets
Max Depth: Maximum degree of each decision tree constructed.
Max Bins: Maximum number of leaf nodes per decision tree (breadth)
Categorical Features Map: Map of features with category set count Eg: 1 -> 10 for feature 10 with values ranging from {0-9}

Parameters:

We initially started exploring by sampling and slicing records in R. We found the most important features out of the core covariates that contribute to the bird in context (Red Winged).
Based on these observations, we eliminated features such as -
“BAILEY_ECOREGION”, “ELEV_GT”, “CAUS_PREC”, “CAUS_SNOW”

We saw weak contributions to the true labels from these features and therefore, felt the need to eliminate these to boost our prediction.

Finally, we selected the following features:

Features	Column Index
BCR	961
ELEV_NED	960
OMERNIK_L3_ECOREGION	963
POP00_SQMI	956
HOUSING_DENSITY	957
HOUSING_PERCENT_VACANT	958
CAUS_TEMP_AVG	964
CAUS_PREC	967
CAUS_SNOW	968
CAUS_TEMP_MIN	965
CAUS_TEMP_MAX	966
NLCDYYYY_FS_TT_7500_PLAND	969-1016
DIST_FROM_FLOWING_FRESH	1091
DIST_IN_FLOWING_FRESH	1092
DIST_FROM_STANDING_FRESH	1093
DIST_IN_STANDING_FRESH	1094
DIST_FROM_WET_VEG_FRESH	1095
DIST_IN_WET_VEG_FRESH	1096
DIST_FROM_FLOWING_BRACKISH	1097
DIST_IN_FLOWING_BRACKISH	1098
DIST_FROM_STANDING_BRACKISH	1099

DIST_IN_STANDING_BRACKISH	1100
DIST_FROM_WET_VEG_BRACKISH	1101
DIST_IN_WET_VEG_BRACKISH	1102

Total features: 71

Pseudocode & Explanation:

Pre-processing steps:

- Update and normalize the “Null/Invalid” values with 0 so that they have no contribution towards the model construction
- Replaced values of label ”X” with 1 because the model is a binary classifier and is expected to predict if the bird was observed(1) or not(0)
- Selected features – 71, 59 of them being the core covariates (as mentioned in the supporting documentation) and the remaining 12 chosen are based on intuition, features that talk about the water bodies in and around the habitat which we feel is an important feature about where specific species could be found

Pseudocode:

1. Read the labeled.csv.bz2 file from input and drop all columns except the above selected ones (72 in total: 71 features and 1 label).

2. Convert it to a LabeledPoint RDD (called parsed) to pass it to the model for training

```
val parsed = input.map { case (k, vs) =>
        LabeledPoint(k.toDouble, Vectors.dense(vs.toArray))
}
```

3. The RDD is then split into _% for training and _% for testing using a seeding factor of _ for randomization of records

```
val splits = parsed.randomSplit(Array(0.8, 0.2), seed = 11L)
val training = splits(0).cache()
val test = splits(1)
```

4. Train the model using the following Tuning Parameters:
    a. # of Trees
    b. Feature subset strategy
    c. Impurity
    d. maxDepth

```
val numClasses = 2
val categoricalFeaturesInfo = Map[Int, Int](6 -> 10,7 -> 10,8 -> 10,59 -> 10,60 -> 10,61 -> 10,62 -> 10,
                                63 -> 10,64 -> 10,65 -> 10,66 -> 10,67 -> 10,68 -> 10,69 -> 10,70 -> 10)
val numTrees = 30
val featureSubsetStrategy = "all"
val impurity = "gini"
val maxDepth = 15
```

```
val maxBins = 32
val model = RandomForest.trainClassifier(training, numClasses, categoricalFeaturesInfo,numTrees, featureSubsetStrategy, impurity, maxDepth, maxBins)
```

5. Predict labels for the testing data using the trained model

```
val labelAndPreds = test.map { point =>
        val prediction = model.predict(point.features)
        (point.label, prediction)
}
val predictionAndLabel = test.map(p => (model.predict(p.features), p.label))
```

6. Save the model to storage for reusability

```
model.save(sc, args(1)+"_model/RandomForest")
val sameModel = RandomForestModel.load(sc,args(1)+"_model/RandomForest")
```

7. Read the unlabeled.csv.bz2 file from code and select additional sampling event id and the corresponding features.(72 in total: 71 features and 1 sampling_event_id).

8. Repeat Step 2.

9. Load the model and use for predicting label for every record in RDD above

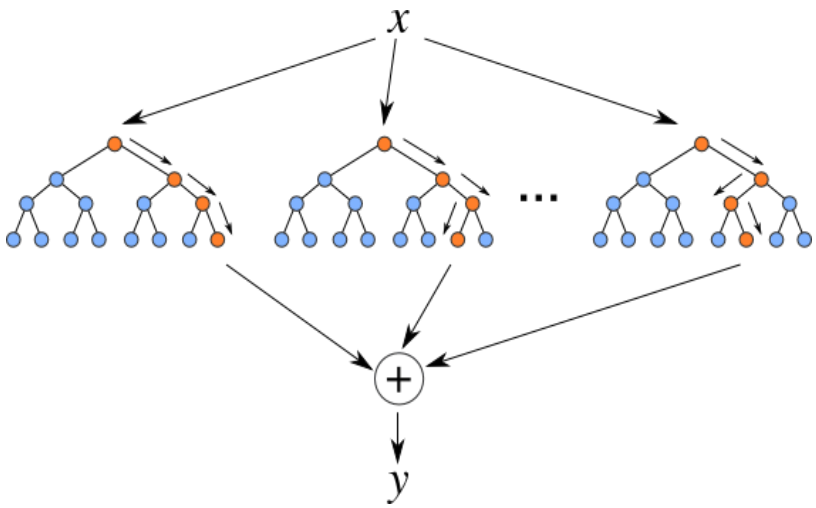
```
val sameModel = GradientBoostedTreesModel.load(sc,args(1)+"_model/RandomForest")
val predictions = unlabeledParsed.map(point => point._1+","+sameModel.predict(point._2).toInt)
```

10. Union the prediction with the event ID RDD and write to output

```
sc.parallelize(headerRDD.union(predictions).collect(),1).saveAsTextFile(args(1))
```

Example:

For all training records (label,(x0….xn)), we create the # of trees as follows:



Using the output predictions, we output the highest occurring prediction for every test and unlabeled record. Finally, we output the predictions in the desired format as-

(SAMPLING_EVENT_ID,SAW_AGELAIUS_PHOENICEUS)

Scaling:

Computation scales approximately linearly in the number of training instances (trees in random forest), in the number of features, and in the maxBins parameter.
Communication scales approximately linearly in the number of features and in maxBins.

Observations (Accuracy of Models Explored):

Number of Trees	Number of Features	Depth of each Tree	Feature subset strategy	Accuracy (%)	Categorial Features Used
10	11	5	auto	72.52	No
10	11	8	auto	73.31	No
10	59	8	auto	73.60	No
10	71	8	auto	74.21	No
15	71	8	auto	75.68	No
25	71	8	auto	75.6	No
20	71	8	auto	75.87	No
20	71	12	auto	77.36	No
30	71	15	sqrt	79.08	Yes
30	71	15	log2	79.09	Yes
30	71	15	all	79.40	Yes

Performance Comparison:

Number of Workers	Steps	Time Taken (seconds)
5	Model Training and Validation	1114
	Prediction	158
10	Model Training and Validation	396
	Prediction	65