Team Members:
Supriya Ayalur Balasubramanian (sayalurb)
Sumeet Mishra (sumish)
Varun Miranda (varmiran)

ASSIGNMENT 4: MACHINE LEARNING

K-NEAREST NEIGHBOURS ALGORITHM

KNN is one of the simplest, non-parametric algorithm, i.e., it doesn't make any assumptions for underlying data distribution. In other words, the model structure is determined from the dataset. KNN is also called Lazy algorithm because it does not need any training data points for generating the model. Instead it uses all the training data used in the testing phase which makes training faster and testing phase slower and costlier in terms of time and memory.

In KNN, K is the number of nearest neighbors which is the core deciding factor. K is generally an odd number if the number of classes is 2. When K=1, then the algorithm is known as the nearest neighbor algorithm. This is the simplest case.

KNN has the following basic steps:

1. Calculate the distance between each training data and all the test data.
2. Find the 'K' closest neighbors.
3. Take a majority vote for the labels.

In the given training and testing datasets, we calculate the distance of each test point with all the training points and pick the 'K' nearest training data points. Then, we sort the array and take the majority voting to assign the label to the test image data. Distance can be calculated using several formulas such as Euclidean distance, Manhattan distance etc.

Explanation of the KNN Code:

Training:

- As KNN algorithm needs the entire training data in the testing phase, we simply copy the entire training data into our model file "nearest_model.txt"

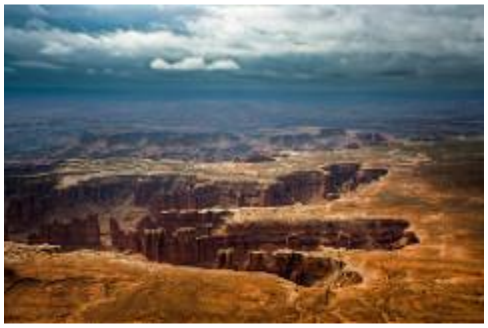
Testing:

- Load the test and model data into NumPy arrays.
- Calculate the **Euclidean** distance for each test data from each of the training image data. Store this distances in a NumPy array of size [36976*943]
- For each column in the NumPy array, take the distance array column and append the training label.
- Sort this distance in increasing order and pick the 'K' smallest distance training points.
- Pick the label (image orientation) which occurs the most among those 'K' neighbors.
- Classify that label, i.e., orientation to the photo_id.
- Repeat the steps 3 to 6 for all the other columns in the NumPy array.

Team Members:
Supriya Ayalur Balasubramanian (sayalurb)
Sumeet Mishra (sumish)
Varun Miranda (varmiran)

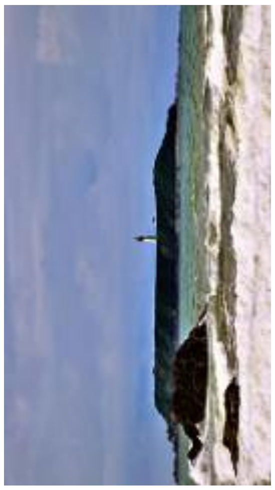
<u>Correctly classified images:</u>

1) test/10099910984.jpg



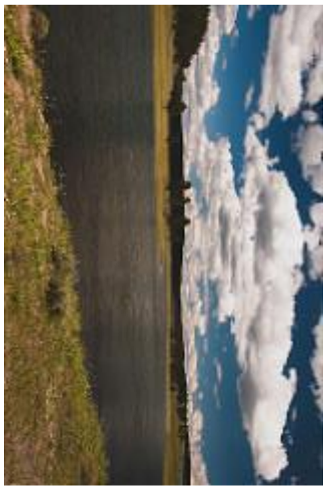
Actual Orientation: 0°
Predicted Orientation: 0°

2) test/10161556064.jpg



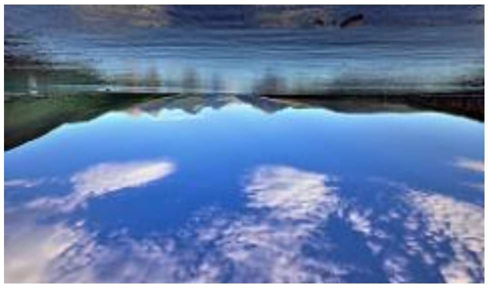
Actual Orientation: 270°
Predicted Orientation: 270°

3) test/ 14018235306.jpg



Actual Orientation: 90°
Predicted Orientation: 90°

4) test/10795283303.jpg



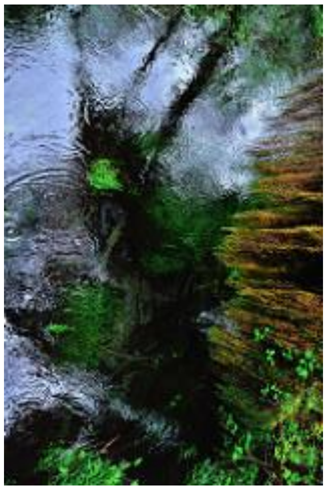
Actual Orientation: 180°
Predicted Orientation: 180°

In all the above sample images, we notice that the images that has sky above and land, green grass or water below are classified correctly.

Team Members:
Supriya Ayalur Balasubramanian (sayalurb)
Sumeet Mishra (sumish)
Varun Miranda (varmiran)

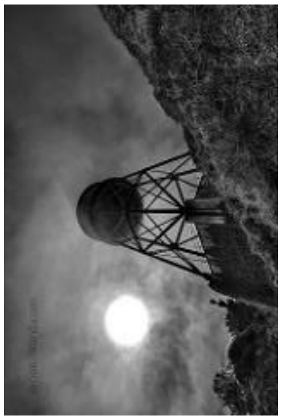
Incorrectly classified images:

1) test/10196604813.jpg



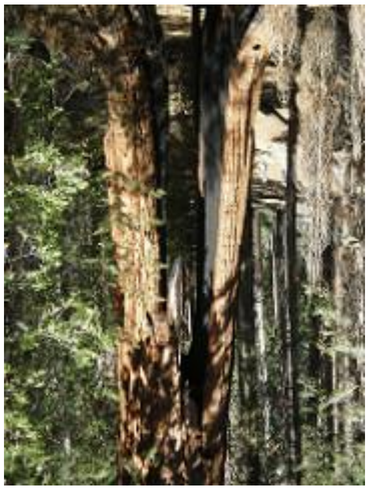
Actual Orientation: 90°
Predicted Orientation: 0°
Explanation: The algorithm predicts 0 degrees for this image. This might be because, in general , a landscape with water will be on the bottom part of the image and hence the algorithm assumes this to be 0 degrees oriented.

2) test/10351347465.jpg



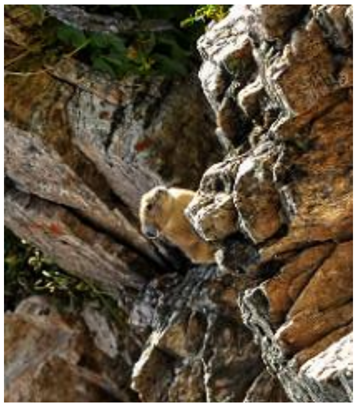
Actual Orientation: 270°
Predicted Orientation: 180°
Explanation: The algorithm predicts 180 degrees for this image. Since the image has only black, white and gray colors, the algorithm isn't able to predict the exact orientation of the image

3) test/1160014608.jpg



Actual Orientation: 180°
Predicted Orientation: 90°
Explanation: The algorithm predicts 90 degrees for this image as this image is symmetrical.

4) test/15227162696.jpg



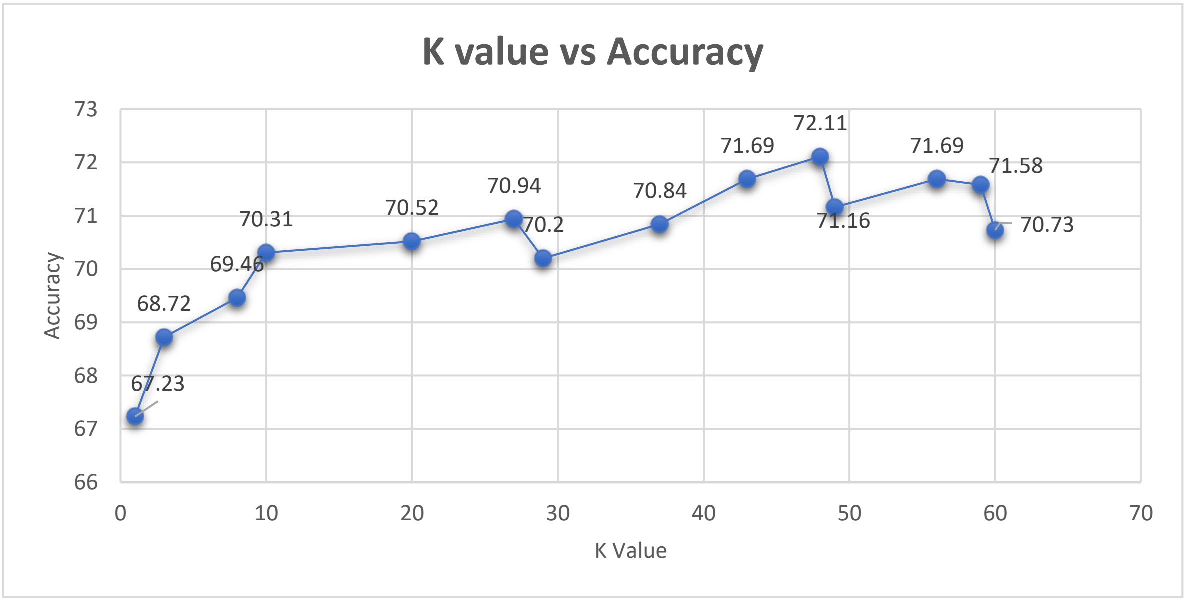
Actual Orientation: 270°
Predicted Orientation: 0°
Explanation: The algorithm predicts 90 degrees for this image as this image is symmetrical and has only one color 'brown' predominantly.

Team Members:
Supriya Ayalur Balasubramanian (sayalurb)
Sumeet Mishra (sumish)
Varun Miranda (varmiran)

Parameter Optimizations:

We tried for different k values to find which k value gives the best accuracy. Below is the table. We find that at k=48 we get the best accuracy and observed no best performance after that.

K Value	Accuracy
1	67.23
3	68.72
8	69.46
10	70.31
20	70.52
27	70.94
29	70.2
37	70.84
43	71.69
48	**72.11**
49	71.16
56	71.69
59	71.58
60	70.73



We would recommend setting k value to 48, for best performance of K neighbors algorithm.

Team Members:
Supriya Ayalur Balasubramanian (sayalurb)
Sumeet Mishra (sumish)
Varun Miranda (varmiran)

ADABOOST ALGORITHM

INTRODUCTION

Adaboost is short form of Adaptive boosting. The idea behind it is constructing a learning algorithms that are combined into a bunch of weak classifiers that take weighted sum that represents the final output of the boosted classifier. The weak classifiers are known as decision stumps.

A typical Adaboost classifier performs binary classification. We can extend this feature for multiclass classification using One vs Many method or One vs One method.

In One vs many, we consider one class to be say 0 degree and the other classes to be anything that is not 0 degree. So basically, we use n number of Adaboost classifiers for n-class problem. In our case, we would need to use 4 Adaboost classifiers in conjunction for classifying an image into either of the four classes (0,90,180,270).

In one vs one, we make Nc2 number of binary classifiers. In this we make classifiers with all possible class combinations, like 0 vs 90, 90 vs 180...etc. Later we assign the class that get majority number of votes that are output by each binary classifier. In case of tie, we randomly pick up one of the class amongst the contenders.

We have implemented One vs One algorithm.

WEAK CLASSIFIER

We follow a simple approach to build a weak classifier based on pixel values. Ideally, we can compare each pixel in our 8*8*3 image with every other pixel. This would require 192^2 comparisons. As this would significantly increase the running time of our algorithm, we randomly pick 500 pair of pixels and compare them with one other.

Let us say for decision Stump 1, we have (c1, c2) pair. If Pixel_Value[c1] > Pixel_Value[c2] in training sample say Tf: Orientation of Tf is the orientation that maximum examples in training file have whenever pixel c1 greater than pixel c2. Else, Orientation of Tf is the orientation that maximum examples in training file have whenever pixel c1 lesser than pixel c2.

For example: say we have c1 = 14, b2 = 100, then we compare if for a training example Ti, value of pixel at 14th position is greater or less than value at 100th position. Let us assume we observe Pixel_Val[14] > Pixel_Val[100]. Suppose there are 1000 training images having Pixel_Val[14]>Pixel_Val[100]. Out of these we see that 600 images have orientation as 0 degrees. Then we assign the training example Ti as 0 degrees.

Note: We use subset of training examples with only the orientation of either class1 or class 2 for a binary adaboost classifier (class1 vs class2).

WEIGHTS

Initially we are going to assign a weight of 1/no. of training examples to each training image. Later whenever a pixel classified with the correct orientation, we reduce its weight. This increases the importance for the wrongly classified images.

PREDICTION OF ORIENTATION FOR TEST IMAGES

The training phase of the algorithm returns a hypothesis containing k pairs of decision stump and their corresponding weights.
For a given test image say Tf, we get predicted orientation using K pairs. Later we compute the total weights. The orientation with maximum total weight is our answer.

For example: Out of 10 decision stumps for image test1.png, 6 predicts it to be 0 degrees and 4 predict it to be 90 degrees. So, we add the corresponding weights of the 6 decision stumps(say 4.2) versus weights of the 6 decision stumps(say 7.5). As the total weight of 90 degrees is more, it gains the weighed majority.

Therefore, we assign Test1.png as 90 degrees.

MULTI-CLASS CLASSIFICATION

As we are using one vs one classifier, we will have 6 classifiers as follows:

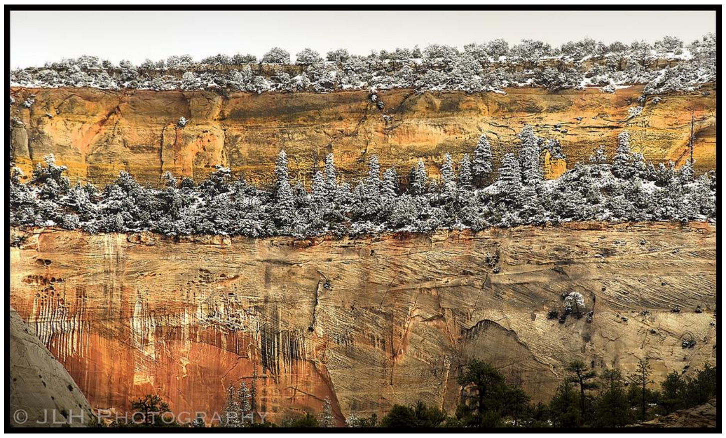
Adaboost(0,90)
Adaboost(0,180)
Adaboost(0,270)
Adaboost(90,180)
Adaboost(90,270)
Adaboost(180,270)

Each classifier predicts one of the two classes. We classify the class that has maximum vote amongst all binary classifiers.

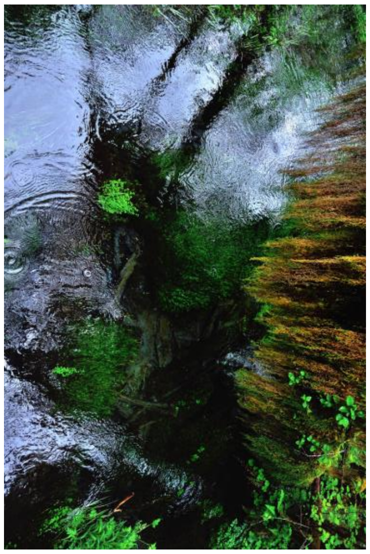
We then store each value of the classifiers above in a dictionary as key value pair and also send this value to model file.

Our test file then picks up our model file and it uses the dictionaries to predict a label for the testing data set.

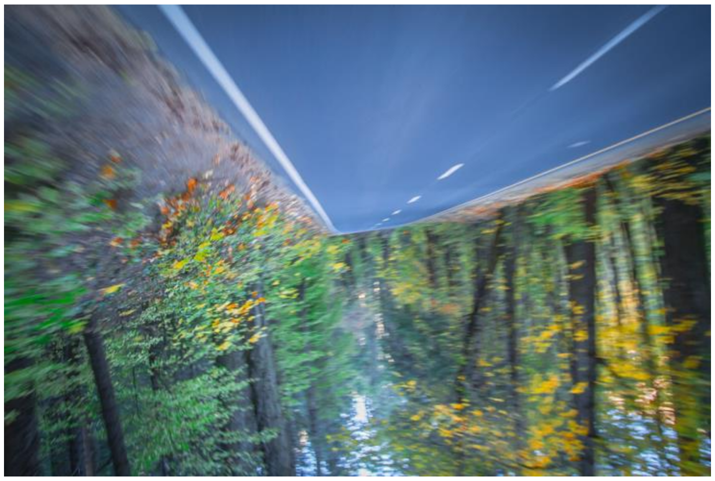
Image Classification



Correct Orientation =0 degree
Our model predicts=0 degree



Correct Orientation =90 degree
Our model predicts=270 degree
This image is taken from top. Our model could not able to predict the correct orientation as the shadows and tree leaves in the water seems to be part of the tree for the model.

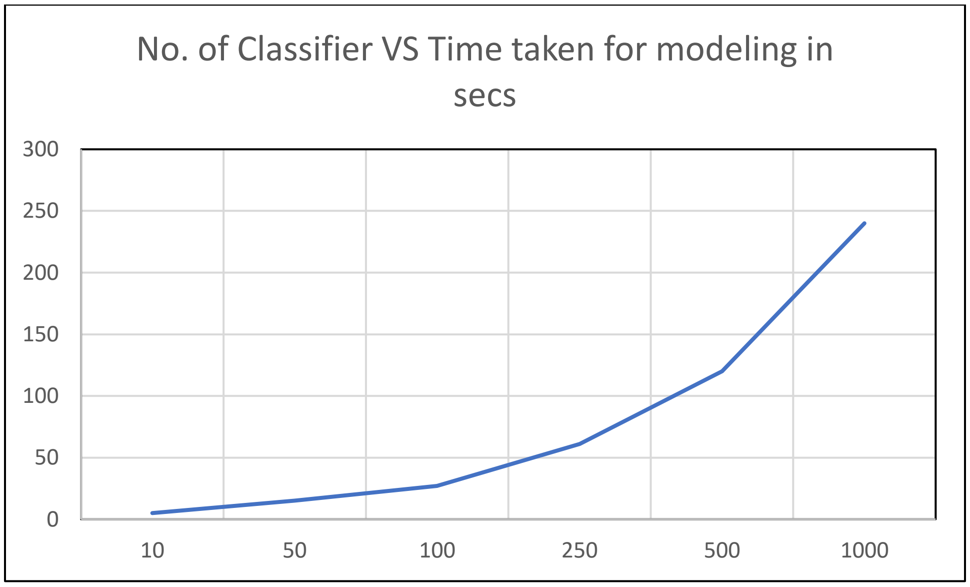


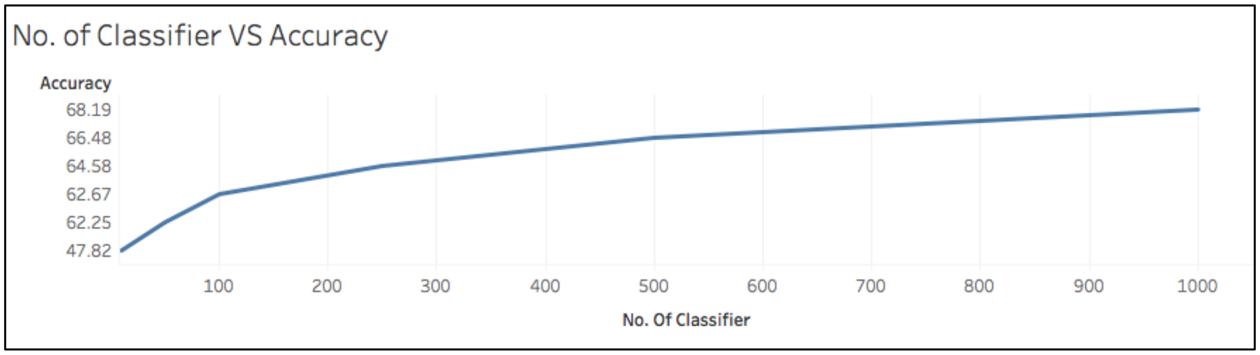
Correct Orientation =180 degree
Our model predicts=0 degree

Our model seems to think the road as sky as the road is mix of blue and white color.

PARAMETER OPTIMIZATION

We observe an increase in accuracy with the increase in the number of classifiers. This is because with more pairs, we would have a better visual information for any image. This would result in better construction of hypothesis for classification.





We observe a significant increase in runtime as well as accuracy with increase in the number of decision stumps.

We would recommend using 500 decision stumps, as we get a good amount of accuracy in a reasonable amount of training time.

RANDOM FOREST

Description of the Code:

First Approach:

1. In order to implement Random Forest, first a Decision Tree needs to be implemented

2. The approach I have tried earlier was to take a random sample of training data (took top 500 rows for checking the decision tree layout)

3. The decision tree function will iterate over various threshold values and the gini index will be calculated for every column

4. The column with the least disorder is chosen as the root node and values of that column > threshold and < threshold will be subsetted and the same process repeats on both the children till the disorder is 0

5. Once the disorder is 0, the process terminates in a leaf node

6. A log is maintained which will have the following entries for instance

ColChosen	Parent_Col	Rule	Threshold	Disorder	Size
3	0	root	0	0.6733903	500
24	3	greater	125	0.18058175	202
99	24	greater	115	0.08903411	111
174	99	greater	130	0.05872	46
192	174	greater	150	0.028	25
121	192	greater	135	0.02238095	19
62	121	greater	150	0.01108571	12
8	62	greater	135	0	5
105	62	less	135	0.002	7
59	105	less	110	0	2
76	121	less	150	0	7
33	174	less	150	0.01514286	21
90	33	greater	115	0.003	14

7. Problem with this approach: Many decision trees will have separate tables like this and imposing a prediction with test data and implementing max voting will take time

<u>Second Approach:</u>

Citation: Implementation of Question, Leaf and Decision Node classes and the general idea from Google Developers

Github Code Link: https://github.com/randomforests/tutorials/blob/master/decision_tree.ipynb

1. The algorithm is devised using the three classes:
Question: that uses all the unique threshold values (0,255) and returns a boolean value if the value in the column is greater than the threshold
Leaf: once the tree terminates in the leaf node, it's frequency of the class variables is computed,
Decision Node: references the question, and its child nodes.

2. The main function of this algorithm is the build tree function, initially the depth of the tree is given as 0, and then it's incremented up to a max depth, beyond which the tree will terminate into a leaf node

3. For splitting the data based on the best threshold value and the best column, the find best split function is used which will call the gini and the information gain functions which determines the best way to split the data. The split is performed by the partition function

4. The function classify takes the test data and performs the rules that it learnt from the training data and predicts the outcome.

5. This entire program is done for one decision tree. Random forest takes multiple decision trees on different samples of training data

6. Max Voting is then done to take the majority of the decisions into account

7. The model file is created using the pickle library: https://docs.python.org/2/library/pickle.html

Model Parameters:

There are three parameters in the model:

1. Number of trees in the forest: There is a tradeoff here between the accuracy and time. The more the number of trees, the more the number of times samples will be generated, and each tree will become an expert at that sample data. Due to this, there is a better chance that the tree will classify correctly for at least majority of the trees

2. Sample Size: The sample size influences the accuracy of the model. If the number of samples is too much, there could be a lot of noise that will be accounted for and the model will take time and if the number of samples is too less, then there is not enough data to train a decision tree

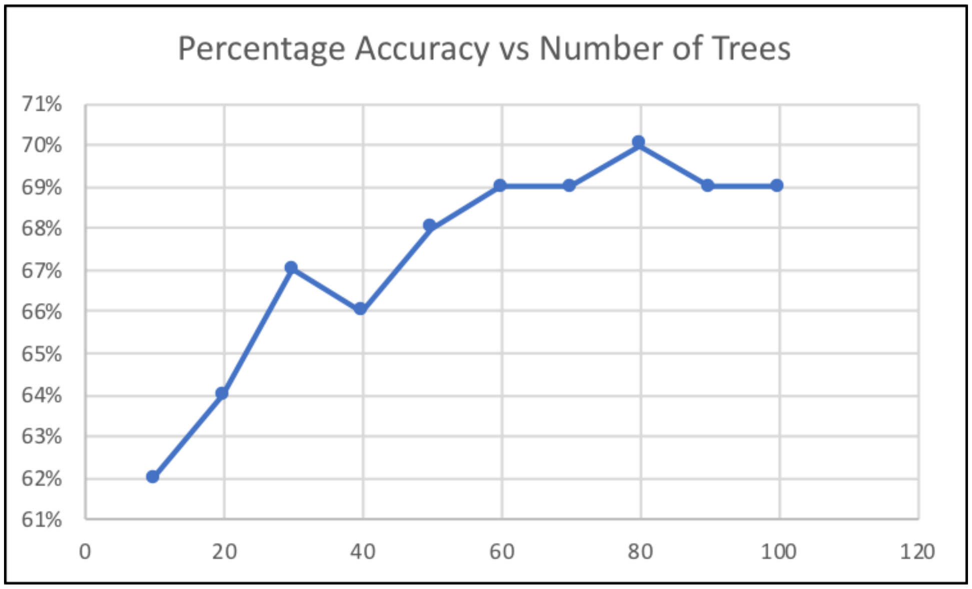
3. Depth: The depth here will ensure that every time a feature and threshold value is selected to split the data on, the split won't necessarily happen till the gini error is 0. The tree will have a fixed depth

The parameters chosen as 60 trees, 100 sample size with random samples, depth fixed at 5 will give an accuracy of 69.88%

Tuning:

Number of trees	Accuracy
10	62%
20	64%
30	67%
40	66%
50	68%
60	69%
70	69%
80	70%
90	69%
100	69%

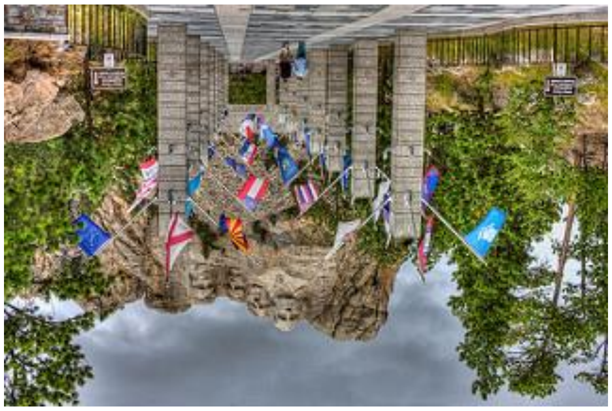
For a fixed sample size of 100 and the depth as 5, the above observations are recorded.



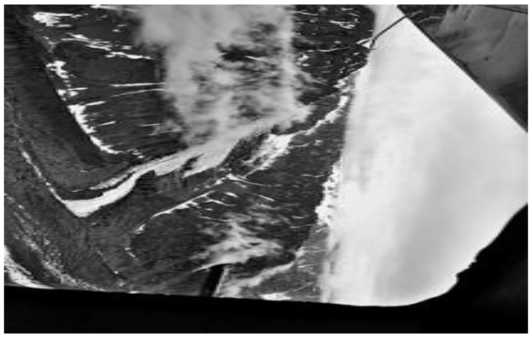
Due to the fact that the accuracy stabilizes after increasing the number of trees beyond 60. The number of trees parameter is fixed at 60 and the sample sizes have been increased from 100. It turns out that even though the sample size is 5000 with 50 trees, the accuracy was 67.55% and the code ran for 8 hours or so. Hence, I would recommend the parameters that I have used to the client (number of trees = 60, sample size = 100, max depth = 5) and the code takes less than 5 minutes to run.

Classification:

<u>Correctly Classified:</u>



Actual: 180
Predicted: 180

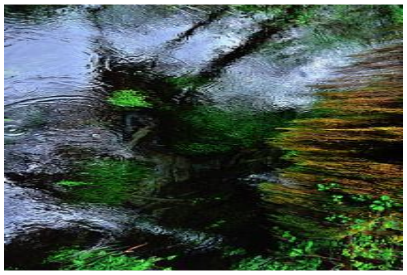


Actual: 90
Predicted: 90

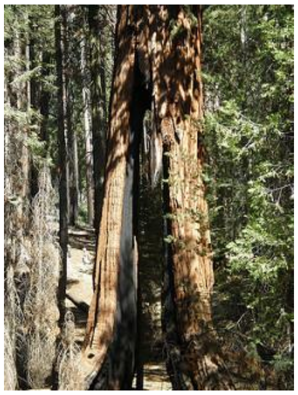
Actual: 270
Predicted: 270

Actual: 0
Predicted: 0

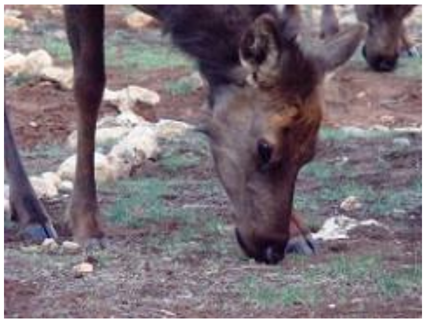
Incorrectly Classified:



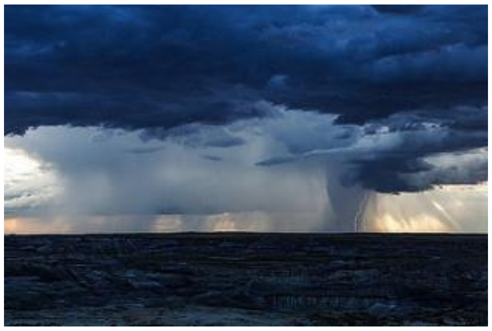
Actual: 90
Predicted: 0
The algorithm predicts 0 degrees for this image. This might be because, in general, a landscape with water will be on the bottom part of the image and hence the algorithm assumes this to be 0 degrees oriented.



Actual: 180
Predicted: 90
The prediction here is 90 degrees because the classifier thought that the leaves look like grass.



Actual: 0
Predicted: 180
The prediction here is 180 degrees as there is a small distinction between the brown color of the deer and the brown color of the ground.



Actual: 0
Predicted: 90
The algorithm predicts 90 degrees for this image. This might be because it’s quite possible that the ground resembles the bark of a tree. If there were no clouds then it would have traced the sky and predicted the correct orientation.

Team Members:
Supriya Ayalur Balasubramanian (sayalurb)
Sumeet Mishra (sumish)
Varun Miranda (varmiran)

BEST ALGORITHM:

We have chosen KNN Algorithm to be our Best model for the given dataset.

PERFORMANCE COMPARISON FOR ALL ALGORITHMS:

We ran all the algorithms for approximately quarter (~9500 images) of the given training set and observed the accuracies as below.

Algorithm accuracy and RUNTIME with half of the dataset

Algorithm	Accuracy	Time taken
KNN	69.5%	2m34.470s
Adaboost	66.2%	1m1.587s
Decision Forest	67.4%	1m40.254s
Best	69.5%	1m34.548s

We observe the accuracy as below when the algorithms are run for the entire training set.

Algorithm accuracy and RUNTIME with full dataset

Algorithm	Accuracy	Time taken
KNN	72.11%	7m41.924s
Adaboost	69.14%	2m2.978s
Decision Forest	69.88%	3m36.838s
Best	72.11%	6m29.538s

We see a decrease in performance when the partial training dataset is used. Since we used partial training dataset, there are less training examples for the algorithm to learn and generate the model. But the computation time is less owing to a smaller number of training examples. Hence, we can conclude that more the training data, better the accuracy of the machine learning algorithm .