**Homework 2: Animal Dataset Analysis**

Name

Institution

Data

1. The two least similar animals are: ('antelope', 'grizzly+bear')

Number of common features: 53

*Code to determine*

# Load animals data as a dictionary

animals\_data = load\_matrix("awa/predicate-matrix-binary.txt")

# Initialize variables to store the least similar animals and the minimum common features

least\_similar\_animals = None

min\_common\_features = float('inf')

# Iterate through all pairs of animals and count common features

for animal1, features1 in animals\_data.items():

    for animal2, features2 in animals\_data.items():

        if animal1 != animal2:

            common\_features = sum(f1 == f2 for f1, f2 in zip(features1, features2))

            if common\_features < min\_common\_features:

                min\_common\_features = common\_features

                least\_similar\_animals = (animal1, animal2)

# Convert animal labels to their corresponding names

animal1\_label, animal2\_label = least\_similar\_animals

animal1\_name = labels[int(animal1\_label)]

animal2\_name = labels[int(animal2\_label)]

# Print the names of the least similar animals and the number of common features

print("The two least similar animals are:", (animal1\_name, animal2\_name))

print("Number of common features:", min\_common\_features)

1. Shortest path (questions): 5

Longest path (questions): 6

Average questions to guess an animal: 5.72

Code to determine

def find\_paths(decision\_tree, animals\_data, node=0, current\_path=[]):

    labels, data = animals\_data

    tree = decision\_tree.tree\_

    left = tree.children\_left[node]

    right = tree.children\_right[node]

    current\_path = current\_path + [node]

    if left == right:  # leaf node

        animal\_label = decision\_tree.classes\_[tree.value[node].argmax()]

        return [(current\_path, animal\_label)]

    else:  # decision node

        left\_paths = find\_paths(decision\_tree, animals\_data, node=left, current\_path=current\_path)

        right\_paths = find\_paths(decision\_tree, animals\_data, node=right, current\_path=current\_path)

        return left\_paths + right\_paths

# After fitting the decision tree

paths = find\_paths(decision\_tree, animals\_data)

shortest\_path = min(paths, key=lambda x: len(x[0]))

longest\_path = max(paths, key=lambda x: len(x[0]))

shortest\_questions = len(shortest\_path[0]) - 1  # Counting internal nodes only

longest\_questions = len(longest\_path[0]) - 1  # Counting internal nodes only

# Total number of animals

total\_animals = len(labels)

# Average number of questions to guess an animal

average\_questions = (sum(len(path[0]) - 1 for path in paths) / total\_animals)

print("Shortest path (questions):", shortest\_questions)

print("Longest path (questions):", longest\_questions)

print("Average questions to guess an animal:", average\_questions)

1. Which question had the most “yes” answers? : oldworld  
   Which had the fewest “yes” answers? : red

*Code to determine*

# Calculate the number of "yes" answers for each feature

yes\_counts = [count(feature\_column) for feature\_column in data.T]

# Find the index of the feature with the most and fewest "yes" answers

most\_yes\_index = numpy.argmax(yes\_counts)

fewest\_yes\_index = numpy.argmin(yes\_counts)

# Get the names of the features

most\_yes\_feature = features[most\_yes\_index]

fewest\_yes\_feature = features[fewest\_yes\_index]

# Print the results

print("Feature with the most 'yes' answers:", most\_yes\_feature)

print("Feature with the fewest 'yes' answers:", fewest\_yes\_feature)

1. Which question was most balanced in terms of “yes” or “no” answers, i.e., where the number of “yes” answers was closest to the number of “no” answers?

“lean”

What is the first question that the decision tree asks at the root node?

“lean”

*Code to determine*

# Calculate the absolute differences between "yes" and "no" answers for each feature

absolute\_diff = [abs(count(data[:, i]) - (N - count(data[:, i]))) for i in range(M)]

# Finding the index of the feature with the smallest absolute difference

balanced\_feature\_index = min(range(M), key=lambda i: absolute\_diff[i])

# Get the name of the balanced feature

balanced\_feature = features[balanced\_feature\_index]

# Print the result

print("Most balanced feature:", balanced\_feature)

# Find the index of the feature used in the root node of the decision tree

root\_feature\_index = numpy.argmin(absolute\_diff)

# Get the name of the feature at the root node

root\_feature = features[root\_feature\_index]

# Print the first question at the root node

print("First question at the root node:", root\_feature)