Project: Avian Asker

Supervising TAs: Liang Wang, Carlos Gonzalez
Base on handout by Noah Jakimo

November 15, 2010

1 Introduction: Guessing Games/Watching Birds

The *Oxford English Dictionary* contains ~500,000 words. Its size is only a little less than 2^{19}, and a binary search can therefore solve the 20 Questions game rather easily. This is why we move to a more interesting and feathery problem.

2 System Requirements: Python 2.6+ (Any Platform)

Avian Asker requires the Python programming language. For tutorial, the respective CS 11 track's website[1] can bring any programmer up to speed.

Feel free to work within the operating system of your choice. Unless that is Windows, you should already have an appropriate version of Python pre-installed. Nonetheless, everyone should check they have a Python version which is 2.6 or higher[2], as well as software that can unpackage the data to get started.

3 Data & Extraction: Caltech-UCSD Birds 200[3]

Images[4]

There are 6,033 images from 200 bird species - listed in classes.txt - within our collection. Kindly ignore the half that test.txt enumerates. Please only work with those in the train.txt file.

Note, our ornithology of JPEG files are catalogued into subdirectories according to species. As you begin bird watching, the script below will display one fledgling at random:

```
import random, Image
infile = open("lists/train.txt","r")
train = infile.readlines()
rndbrd = train[random.randint(1,len(train))]
Image.open('images/'+rndbrd.rstrip()).show()
```

[1] http://www.cs.caltech.edu/courses/cs11/material/python/
[2] http://www.python.org/download/
[3] Welinder P., Branson S., Mita T., Wah C., Schroff F., Belongie S., Perona, P. "Caltech-UCSD Birds 200". California Institute of Technology. CNS-TR-2010-001. 2010.
[4] http://www.vision.caltech.edu/visipedia-data/CUB-200/images.tgz

Attribute Annotations[5]

Each time an image was viewed, 1 of 1,577 MTurk participants provided 5 of 288 attribute responses. All possible attributes are listed in attributes.txt and all responses have been logged in labels.txt (Please refer to README.txt for their format). Labels may be collected into a Python list as follows:

```
labels = []
infile = open("labels.txt","r")
for line in infile.readlines():

   entry = map(int, line.split())
   labels.append(entry)
```

The file is large (almost 9 million lines), and constantly searching through it will surely waste your time. Instead, consider building Python dictionaries while you read in, and saving those with the `cpickle` Python module.

4 The Interface: You ask, we answer (Note there are changes!)

You will write a method, `myAvianAsker`, that accepts two arguments, the first is a string with the path to an image file, and the second is a list of questions and answers. Your function should return the index of either an attribute to ask a question about, or a bird species.

Species are numbered from 1 to 200 as specified in the file `specie_name.txt` that you used for the milestone, or if needed you can find these again in the file `student_image.txt`, and attributes are numbered 0 to 287 in `attributes.txt`. Your function should return 0 to 287 if it is asking about this attribute, or 288 to 487 if it is guessing the number of the species (simply add 287 to the species number). Your function can guess incorrectly, and our program will respond with a 'no' and will keep on asking for more questions.

This time our function returns a list with two elements. The first element is the one you are familiar with, which is an answer to your question. These are encoded as 0-"no", 1-"yes" or 2 -"don't know". Note here that you cannot assume you will always obtain a yes/no answer as in the milestone. There may be some attributes for which no answer is provided in the data set.

The second element that will be returned determines the certainty of the answer: "0" for certain, "1" for probable, and "2" indicating the answer was just a guess. For the exact file, please download `gameplay.py` in `aa_final.zip`. Note that this file just simulates testing your code with a number of species from the same training set you are using. The real evaluation will also test your code against unseen data.

```
QAs = []
while True:
         Q = myAvianAsker(imagename, QAs)
         if Q-nattributes+1 == rndbrd:
                  print("You have guessed correctly, the bird is "...\n")
                  break
         elif Q >= nattributes + nspecies:
                print("The question is out of range")
                continue
         elif Q >= nattributes and Q != rndbrd:
                  A = 0  #the answer is wrong
         else:
                  A = #answer to your question in the format [a, b] where a,b=0,1,2
QAs.append([Q, A])
```

[5] http://www.vision.caltech.edu/visipedia-data/CUB-200/attributes.tgz

5 Goal: Noisy Classification on a Withheld Data Set

Ultimately, we will return our attention to the data in `labels.txt` Your Avian Asker will then have to handle conflicting attribute responses and use the fewest questions to identify a bird's species. However, you will prepare your Avian Asker with a subset of the `labels.txt`.

You are ONLY allowed to used the subset given to you for training. The remaining portion of the data will be used for testing the performance of your Avian Asker, which will be previously unseen data for your algorithm. Therefore, as you have access to `lables.txt` we ask you to refrain from using this file for training your Avian Asker. Doing so will be consider as cheating, and the Honor Code is in effect. Note that testing your code against unseen data is the ultimate goal as you want your algorithm to work well with any new bird image it encounters (or at least any new image of any of the 200 species considered in this data set). We will also test your code against the training data for completeness, but a bigger weight will be assigned to your performance on the withheld test set.

Please download the archive `aa_final.zip` where you will find the following files:

- `/student/student_dataset.txt` This is the one and only file you will use that contains answers to the various attributes of each bird. In this case, there are many responses for the same attribute, for each species, and the answers may or may not be conflicting. The answers are also qualified to be certain, probable, or simple guesses. Also take into account that there might not be an answer to every single attribute for each bird. As you may have already noted, the format of this file is

  `<image_id> <attribute_id> <is_present> <certainty_id> <worker_id>`

- `/student/student_images.txt`: This file relates the image id to the species, as well as to the path to the image. The specie index can be obtained from this file as well
- `/student/student_cPickle.txt`: This is a cPickle file that you need to run the gameplay and stores all of the training data in the dictionaries we use later. In this case, using this module is crucial, otherwise reading every time millions of lines of data will waste your time.
- `gameplay.py`: This python file runs the Q & A game described before. A sample `myAvianAsker` function is provided which asks questions chosen at random and makes random guesses, so that you can see how your function should work. However you should define your `myAvianAsker` inside this file. See section 7 for instructions of how to submit your function.
- `/student/attributes.txt`: This file describes each of the 288 attributes you can ask questions about.

6 Computer Vision Possibilites

For this final version, you are allowed to use computer vision algorithms to help you gain information about the bird and reduce the number of questions you need to ask. This is why your function has a filename as an argument. This filename corresponds to the path of an image of the 200 bird images data set. If you have not already done so, please download all images at

`http://www.vision.caltech.edu/visipedia-data/CUB-200/images.tgz`

To use this images efficiently, you are encouraged to used the annotations provided for each bird. This correspond to `.mat` files that provide a mask to the bird image. The mask is made of black and white pixels, where the white pixels correspond to the area in the image where the bird lies in. Download these annotations and some Matlab sample code that plots an image and its annotation so that you can see this in action. A sample run will show you an image like the one below.

`http://www.vision.caltech.edu/visipedia-data/CUB-200/annotations.tgz`

As it may be obvious already, parsing the filename of the image gives the corresponding species name. Using this during the QA session is again is considered as cheating!

7 Performance

The performance of your algorithm will be scored based on the number of questions needed in average to find the correct bird for previously unseen images. A small weight will also be given to the performance of the algorithm on the training set.

8 What to Turn In

You should turn via email to the TAs in charge two files:

- A short description of your algorithm in English. One or two pages should be enough.
- A python file that contains your code. Your file should be named `AvianAsker_A##.py` where `##` corresponds to your group number.

  Your code should come in one of two possible forms. No other forms will be accepted.

  - A class with the above name, that includes a method named `myAvianAsker` and a method `init`. As you noted from the Milestone, those who implemented classes may have had problems running their code sequentially, *i.e.* when tested in a loop for many species.

    To solve this problem, we will do the following. First, create an object of your class outside the loop that tests sequentially your code. This will only be carried out once for efficiency purposes. Probably here you will want to create your data structures. Once the for loop starts, we will call the initialization function `init` that you will write, which serves the purpose of carrying out any initialization or re-initialization needed, before the QA game starts over again. Finally, inside the QA game, we will call your `myAvianAsker` function.

    In `gameplay.py` there are a few commented lines that show you how your code will be called and initialized as described above.
  - A file with the name above that defines the `myAvianAsker` function. That is, your code should be tested by simply importing this file to our test code and calling your function.

For this final submissions we will NOT accept codes that need to be "pasted" into our evaluation code, and NO re-submissions will be accepted. So make sure your code does not crash before submitting.